Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

APPROXIMATE STRING MATCHING METHODS FOR DUPLICATE DETECTION AND CLUSTERING TASKS

by

Oleksandr Rudniy

Approximate string matching methods are utilized by a vast number of duplicate detection and clustering applications in various knowledge domains. The application area is expected to grow due to the recent significant increase in the amount of digital data and knowledge sources. Despite the large number of existing string similarity metrics, there is a need for more precise approximate string matching methods to improve the efficiency of computer-driven data processing, thus decreasing labor-intensive human involvement.

This work introduces a family of novel string similarity methods, which outperform a number of effective well-known and widely used string similarity functions. The new algorithms are designed to overcome the most common problem of the existing methods which is the lack of context sensitivity.

In this evaluation, the Longest Approximately Common Prefix (LACP) method achieved the highest values of average precision and maximum F_1 on three out of four medical informatics datasets used. The LACP demonstrated the lowest execution time ensured by the linear computational complexity within the set of evaluated algorithms. An online interactive spell checker of biomedical terms was developed based on the LACP method. The main goal of the spell checker was to evaluate the LACP method's ability to make it possible to estimate the similarity of resulting sets at a glance.

The Shortest Path Edit Distance (SPED) outperformed all evaluated similarity functions and gained the highest possible values of the average precision and maximum

F_1 measures on the bioinformatics datasets. The SPED design was inspired by the preceding work on the Markov Random Field Edit Distance (MRFED). The SPED eradicates two shortcomings of the MRFED, which are prolonged execution time and moderate performance.

Four modifications of the Histogram Difference (HD) method demonstrated the best performance on the majority of the life and social sciences data sources used in the experiments. The modifications of the HD algorithm were achieved using several re-scorers: HD with Normalized Smith-Waterman Re-scorer, HD with TFIDF and Jaccard re-scorers, HD with the Longest Common Prefix and TFIDF re-scorers, and HD with the Unweighted Longest Common Prefix Re-scorer.

Another contribution of this dissertation includes the extensive analysis of the string similarity methods evaluation for duplicate detection and clustering tasks on the life and social sciences, bioinformatics, and medical informatics domains. The experimental results are illustrated with precision-recall charts and a number of tables presenting the average precision, maximum F_1, and execution time.

APPROXIMATE STRING MATCHING METHODS FOR DUPLICATE DETECTION AND CLUSTERING TASKS

by
Oleksandr Rudniy

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

January 2012

APPROVAL PAGE

APPROXIMATE STRING MATCHING METHODS
FOR DUPLICATE DETECTION AND CLUSTERING TASKS

Oleksandr Rudniy

Dr. James Geller, Dissertation Co-Advisor — Date
Professor of Computer Science, NJIT

Dr. Min Song, Dissertation Co-Advisor — Date
Assistant Professor of Information Systems, NJIT

Dr. Narain Gehani, Committee Member — Date
Professor of Computer Science, NJIT
Dean of College of Computing Sciences, NJIT

Dr. Vincent Oria, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Xiaohua Tony Hu, Committee Member — Date
Associate Professor of Drexel University, Philadelphia, PA

BIOGRAPHICAL SKETCH

Author: Oleksandr Rudniy

Degree: Doctor of Philosophy

Date: January 2012

Undergraduate and Graduate Education:

- Ph.D. Student in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, August 2011

- Master of Science in Applied Mathematics,
  Kharkiv State University of Radio Electronics, Kharkiv, Ukraine, 2001

- Bachelor of Science in Applied Mathematics,
  Kharkiv State University of Radio Electronics, Kharkiv, Ukraine, 2000

Presentations and Publications:

M. Song and A. Rudniy, "Detecting duplicate biological entities using Markov random field-based edit distance," *Knowledge and Inform. Syst.*, vol. 25, no. 2, pp. 371-387, 2010.

A. Rudniy et al., "Detecting duplicate biological entities using Shortest Path Edit Distance," *Int. J. of Data Mining and Bioinf.*, vol. 4, no. 4, pp. 395-410, 2010.

A. Rudniy et al., "Shortest Path Edit Distance for Enhancing UMLS Integration and Audit," in *Proc. of AMIA 2010 Symp.*, Washington, D.C., 2010, pp. 697-701.

A. Rudniy et al., "Shortest Path Edit Distance for Detecting Duplicate Biological Entities," in *Proc. of ACM Int. Conf. on Bioinf. and Comput. Biol.*, Niagara Falls, NY, 2010, pp. 442-444.

M. Song and A. Rudniy, "Detecting Duplicate Biological Entities Using Markov Random Field-Based Edit Distance," in *IEEE Int. Conf. on Bioinf. and Biomed.*, Philadelphia, PA, 2008, pp. 457-460.

A. Rudniy et al., "Histogram Difference String Distance for Enhancing Ontology Integration in Bioinformatics," submitted to *Int. Conf. on Bioinf. and Comp. Biol.*

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF TABLES
(Continued)

Table **Page**

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

The Approximate String Matching (ASM) problem emerged in the 1960s and evolved in the subsequent decades in a number of different fields. Research efforts in this area were motivated by problems of bioinformatics, signal processing, and text retrieval [65]. Today, research continues in the above-mentioned domains, although the ASM application area has been extended into new fields such as medical informatics, human speech research, and others. Thus, development of new, more effective methods for approximate string matching remains an important research task.

1.1.1 Bioinformatics Applications

Exact string matching is of little use for bioinformatics, since exact pattern matches are very rare [65]. Thus ASM is essential to many applications in this field. ASM methods are used to find common motifs or similarities in DNA, RNA, or protein sequences, where differences are caused by mutations or evolutionary alterations [67]. Common motifs among RNA chains can provide information on RNA functionality and help to classify RNA families [68]. A crucial sub-problem of reconstruction of phylogenetic trees is solved by applying ASM methods to find genetic sequence alignment. Recent applications of ASM are in solving problems of structure matching and in discovering unknown patterns in bioinformatics [65].

The bioinformatics field has experienced tremendous growth, which subsequently established a massive volume of data available from multiple specialized databases. These sources contain diverse information including annotated biological sequences, three-dimensional molecular structures, and genetic and

physical maps [69]. Various ASM methods are used by scientists to solve current research tasks such as retrieving sequences from existing databases that are homologous to newly discovered ones, and establishing multiple sequence alignment to discover similarity patterns to predict the function, structure, and evolutionary history of biological sequences [69].

1.1.2 Medical Informatics Applications

Significant expansion of medical data sources within heterogeneous healthcare information systems has resulted in a redundant and sometimes inaccurate information split among multiple databases. This phenomenon has caused a problem of record linkage and duplicate detection in medical databases. Research tasks include patient record aggregation from multiple databases based on a minimum profile (i.e., a set of features such as last name, first name, gender and date of birth) [78] and term matching for source integration, auditing, and biomedical data mining applications. The latter task is considered in the context of the Unified Medical Language System (UMLS), a well-known, long-term research project [79]. ASM methods are used for adding to, updating, or auditing UMLS vocabularies. ASM methodologies are also important for facilitating biomedical information extraction, fact finding, relationship search, and concept discovery [80].

1.1.3 Human Speech Applications

A very active area for sequence comparison methods is speech research [81]. Speech recognition is one of the well-known application domains of ASM techniques. The seminal task has been to transform spoken information into textual data. Recent research directions in the speech recognition field are the transcription of speech data from novel data sources such as multimedia Web repositories, spoken language

comprehension research on children less than ten years of age, delivery of accurate speech transcripts automatically retrieved under diverse circumstances, cross-lingual language modeling, self-adaptive language machine learning, and others [70].

1.1.4 Signal Processing Applications

Information coded by strings of symbols is often transferred over noisy channels such as radio, telegraph, microwave transmission, etc. [81]. The noise adds errors into the received signal, thus introducing a problem of detection and correction of errors.

The classical Levenshtein edit distance appeared as a result of research work aiming at solving the problem of correction of errors introduced into data transmitted over physical channels [20]. The Levenshtein distance function and its modifications play a major role in sequence comparison [81]. Signal processing theory is used to estimate such errors. The goal of the field is to recover as much of the original transmission data as possible [65].

1.1.5 Text Retrieval Applications

Optical Character Recognition (OCR) is the most widely applied technology for utilization of retrospective documents [72]. OCR allows conversion of printed documents into electronic form for subsequent database storage and indexing. Inevitably, during OCR processing, a number of errors are introduced into the original text. This leads to search failures for finding exact matches by a search query [71]. ASM is one of the key tools for handling OCR errors [72]. Post-editing and text processing to correct errors can be made less labor intensive and more efficient by using ASM.

Another ASM application in the text retrieval domain is record linkage. The record linkage process involves finding information that refers to the same entity

when no unique identifier is available. A unique identifier may be absent because it was impossible to collect, because of confidentiality procedures, or because of introduced data errors. This research area refers to the problem of duplicate detection in a single source or linking records among multiple sources. Examples of datasets for this task are census data and business or personal listings, where the same entity may be listed in multiple categories or where the same entity's record fields are coded differently in multiple sources [82]. Other popular ASM applications are spell checkers, natural-language interfaces, computer-aided tutoring programs, language-learning software, spoken text retrieval, handwritten text recognition, and others [65, 72].

1.1.6 Musical Data Mining Applications

Content-based music access and retrieval is still in its early stages of development [73]. Musical pattern extraction is used in music generation, retrieval and analysis [74]. Unfortunately, exact string matching is of little use in music retrieval. ASM methodologies are applied to music notations consisting of note sequences with associated pitch and rhythm to solve the following tasks: match complete melodies, locate a fragment in a melody database, and retrieve melodies from musical databases [75, 76]. Other applications of string distance methods in musical data mining are identification of musical pieces, copyright infringement detection, musicological analysis, and singing tutorship.

1.1.7 Bird Vocalization Applications

Research on bird songs experienced dramatic growth in recent years. It is important because songs are the main means of communication among birds. For some species, bird songs are inherited from generation to generation and have dialect-like variations

depending on geographical location [81]. These features are rare among non-primates and are similar to human language, which makes it an important research topic. Since bird songs are coded as sequences of string characters, ASM methods are applied to compute distances between them.

1.1.8 Document Clustering Applications

String similarity functions are widely used in document clustering applications [99]. Clustering refers to the problem of dividing documents or records into joint or disjoint sets possessing similar characteristics [98]. The entities assigned to the same cluster are similar to each other according to a certain criterion while the documents assigned to different clusters are dissimilar. String similarity metrics may be used as criteria functions in the clustering task [101]. Clustering is different from the classification problem since the number of clusters, their properties, and their composition are not known in advance [100].

Cluster analysis is a technique that allows the identification of groups, or clusters, of similar objects in multi-dimensional space [117]. It was initially introduced in the field of information retrieval for improving the efficiency of the serial search task [118]. It has become increasingly common to apply clustering to large databases due to growing volumes of collected data of all sorts [102, 103]. Document clustering has applications in a number of fields.

Within the field of data mining, clustering methods are used in database segmentation, in predictive modeling, and in the visualization of large datasets [103]. Also, clustering is often used as a pre-processing step inside of a larger data mining application [107]. Multi-document summarization through the discovery of topic hierarchies and the grouping of duplicate or nearly-duplicate documents are among other clustering applications in data mining [119].

The number of electronic documents published on the Internet and on corporate networks has experienced tremendous growth in recent years [104, 105]. The World Wide Web is often characterized as a large unstructured database, which is subject to a number of document clustering applications [108, 109]. In addition, colossal amounts of textual information are stored within archives of scientific and technical publishing houses and of media companies. These areas are of great practical interest for clustering applications [106]. Clustering provides a way to organize a large collection of unstructured, unlabeled hypertext documents into labeled categories that can be discriminated and disambiguated from one other [113].

Clustering Web search engines have recently gained popularity. These applications group the returned search results into a hierarchy of labeled clusters. This improves on typical engines that apply clustering methods to results after they are returned by a well-known meta search engine [111]. To summarize, document clustering in the World Wide Web domain consists of the following tasks: clustering retrieved documents for better presentation, clustering documents in digital libraries, developing automated document taxonomies, and retrieving cluster-oriented information more efficiently [114]. A more recent Web-related clustering application to emerge is unwanted email (spam) detection [120].

Medical informatics and bioinformatics widely employ clustering methods for data mining tasks. It is possible to find functionally related genes and proteins and classify them by previously unknown roles after grouping the genetic data into clusters [110]. Clustering is efficiently applied in micro array analysis to identify potential local patterns within genes and to help discover macroscopic phenotypes of related samples or previously undetectable biological cellular processes of genes

[112]. Clustering and segmentation techniques are used to split DNA and protein sequences into modules that can be assigned specific molecular functions [115].

ASM methods are extensively used to assign term weights in clustering applications. Clustering methods incorporate similarity metrics such as TFIDF, cosine similarity, Jaccard function, Levenshtein, Needleman-Wunsch distance, and others [115, 116]. Since the uses of these metrics are constantly growing, there is increased demand for improvement of existing ASM metrics used for clustering tasks.

1.2 Organization of the Dissertation

Chapter 1, Introduction, presents the motivation behind this research and analyzes past and current ASM applications in various domains. Chapter 2, Background, reviews related work in the field of ASM, defines the research problems, presents the theory relevant to these problems, and states the formulas of well-known string similarity metrics. Chapter 3, Data Sources, describes in detail the data sources selected for the evaluation. Chapter 3 concludes with the presentation of the standard methods used in information retrieval for string similarity function evaluation. Chapter 4, Similarity Functions for Duplicate Detection and Clustering Tasks, introduces the string similarity algorithms developed by the author and provides detailed specifications. Chapter 5, Evaluation, begins with the description of an experimental test bed, the open-source Java toolkit known as Second String [28]. The chapter includes a detailed discussion of the methodology used to build the experiments. Chapter 5 continues with a thorough presentation of the results obtained during the numerous performed experiments, accompanied by supporting charts and tables. Chapter 6, Summary, concludes the thesis with a discussion of the accomplished work, conclusions, and directions for future work.

The work presented in Section 4 was published before or submitted for publication. The work related to the Markov Random Field Edit Distance (MRFED) method described in Section 4.1 was published in [49, 50]. The research in Section 4.2 concentrating on the SPED method was published in [96, 123, 124]. The material in Section 4.3 and Section 4.4 was submitted for publication.

CHAPTER 2

BACKGROUND

2.1 Summary

String distance metrics constitute the central part of approximate string matching methodologies. Decades of research efforts in this field have produced a vast number of algorithms applied to many scientific problems. Nevertheless, due to the dramatic growth of electronic knowledge sources there is a need for more efficient algorithms capable of solving newly arising problems.

This work introduces a family of novel string similarity methods, which outperform a number of effective well-known and widely used string similarity functions. The new algorithms are designed to overcome a common problem of the existing methods, namely, the lack of context sensitivity.

2.2 Review of the Related Work

The task of approximate string matching in its most general form is to measure similarity or dissimilarity between two strings. Several modifications of non-exact string matching methods that allow errors exist in bioinformatics, medical informatics, data mining, signal processing, text retrieval, optical character recognition, file comparison, image compression, handwriting recognition, virus and intrusion detection, and in many other fields. Depending on the knowledge domain, these methods may have different final goals, e.g., to link records from separate datasets or to find a misspelled word in a text. Nevertheless, all of these tasks can be accomplished using string distance functions, which are called edit distances, distance metrics, string comparators, or string similarity metrics.

Research on ASM began in the 1960s, and it continues up to the present. The classic case of the edit distance was introduced by Levenshtein in 1966 [20]. This algorithm remains popular today due to its extension by Wagner and Fischer [21]. The latter work allowed a string to be composed of any finite alphabet rather than the binary one used in Levenshtein's approach [20]. The binary reverse operation was converted to the substitution of a character by another character; variable weight assignment was allowed for individual edit operations; and a dynamic programming algorithm was provided to calculate the distance between two strings.

Since that extension, numerous research efforts have been made to improve existing string similarity techniques, to adapt the algorithms to new fields of application, or to introduce a completely new approach for the same task. Fellegi and Sunter [14] extended their theory of record linkage by presenting a formal model. Its core idea is to use the relative frequencies of strings being compared, e.g., a rare word found in pairs of records taken from two files gains more weight than a word that is used frequently. Many later applications of frequency-based string matching used Fellegi and Sunter's work with additional adjustments.

A dynamic programming algorithm introduced by Needleman and Wunsch [22] is considered to be the first technique that could find the global alignment between two amino acid sequences. Eleven years later, another famous project conducted by Smith and Waterman [23] targeted the local alignment problem, which emerged as the result of the growth in molecular data research. Their paper proposed a different method of assigning similarity scores in order to find the optimum local alignment of sub-sequences at the expense of the global score. The worst-case complexity of this method is of the order of the string lengths product. A more efficient solution was introduced by Ukkonen in 1985 [24] with the complexity

proportional to the product of the longer string relative to the edit distance value for these strings.

In 1993, Marzal and Vidal [25] demonstrated that the normalized edit distance significantly improved the Levenshtein Distance. Marzal and Vidal stressed that in order to get better results, the normalization should be performed within a dynamic programming algorithm. Marzal and Vidal stressed that the post-normalization of a distance metric produces worse results compared to their method. The post-normalization consists of two steps: first, a not-normalized distance between two strings is computed, and then it is divided by the length of its edit path.

Superior results were obtained in the record-linkage domain by applying variants of the Jaro metric [26], which is based on the number and order of the common characters in two strings. Winkler was able to further improve this algorithm by introducing a re-scorer, which adjusts the final value of the function depending on the length of the longest common prefix [27].

Cohen et al. [28] designed the Soft TFIDF technique, which extended the TFIDF method by adding similar tokens in addition to equal tokens. The experiments described in their paper showed promising results for matching datasets.

A supervised learning method based on data training was shown to outperform several selected metrics by Bilenko et al. [29]. Bilenko et al. also experimented with hybrid methods by combining several distance metrics.

The problem of identifying duplicate records in databases was originally identified by Newcombe [12] as record linkage. Newcombe's study [12] associated a birth record with a marriage record from different databases when information in both cases pointed to the same couple. Positive weights were associated with matching

fields such as name, place of birth, and age. Mostly, the total record weight was sufficient to decide whether a pair was a match or a non-match.

In bioinformatics, data cleaning and integration are relatively young problems. Recently, they have received more attention in the areas of effective discovery and management of biological entities. The Freely Extensible Biomedical Record Linkage (Febrl) system is the first major duplicate detection system in biological databases [13]. Febrl employs probabilistic data cleaning and standardizing based on Hidden Markov Models. In addition, it offers probabilistic data linkage based on the classical Fellegi and Sunter model [14].

Another work utilizing Markov's ideas was conducted by Singla and Domingos [15]. In this work, various entity matching algorithms such as TFIDF and Winkler are combined. As shown in their work, the strength of using Markov logic is its flexibility in attaching weights to first-order formulas and in viewing them as templates for features of Markov networks [15]. However, Singla and Domingos did not incorporate edit distance algorithms into Markov logic, whereas the authors' MRF-based approach does.

An additional major study by Tsuruoka et al. [16] introduces logistic regression for learning a string similarity measure from a gene/protein name dictionary. They use both synonymous pairs of strings and non-synonymous pairs when optimizing the similarity measure. The experimental results show that using diverse types of information improved the accuracy of detecting similar gene/protein names.

Koh et al. [4] examine a duplicate detection problem applied to biological databases. With selected matching criteria, they compute similarity scores for corresponding fields of known duplicate pairs, generate association rules, and detect

duplicates by using heuristic rules. In the work by Koh et al. [4], performed experiments demonstrate that association rules achieve higher efficiency in duplicate detection compared to user-defined rules.

Popular in bioinformatics, the Smith-Waterman distance [17] was designed to find optimal alignments between biological sequences, such as DNA and proteins. It is based on a dynamic programming approach and allows gaps as well as character-specific match scores. The Smith-Waterman algorithm is widely used to perform local sequence alignment.

Research by Herbert et al. [18] describes a toolkit, BIO-AJAX, designed to improve biological data quality by eliminating duplicate entries in protein repositories using various customized database operations. Although BIO-AJAX requires some initial supervised learning, later the tool runs without curator interaction. In the experimental section of the work [18], Herbert et al. demonstrate how the toolkit can solve a nomenclature problem in a phylogenetic and evolutionary system, TreeBASE.

One additional integrated framework extends the line of record linkage applications in the domain of bioinformatics. KitEGA was proposed to evaluate grouping techniques for biological data [19]. This approach treats duplicate detection as a grouping task where grouped data entries represent the same entity. Bauckmann [9] addresses two issues regarding the integration of biological databases: 1) detecting intra-schema relationships and 2) detecting inter-schema relationships. In order to resolve these issues, the SPIDER algorithm was proposed for detecting inclusion dependencies (INDs) as a precondition for foreign keys. The performance of SPIDER was tested with three protein databases: UniProt, SCOP, and PDB. Bauckmann [9] demonstrates how duplicate detection can be used for integrating protein databases that have different schemas.

The main problem of existing edit distance algorithms is that match decisions are made independently for each candidate pair [30, 31]. Thus the contextual dependencies between adjacent characters are neglected.

To resolve this context-free problem of the edit distance, this research introduces several novel string similarity methods, achieving context dependency in different ways. The MRFED method is the adaptation of the Markov Random Fields (MRF) concept [32], successfully used in image recognition and computer vision, to the string matching domain. The MRFED method exploits the notion of the neighborhood system, cliques and clique potentials from MRF theory in the context of the Needleman-Wunsch distance [22]. The Shortest Path Edit Distance (SPED) incorporates substring match operations to achieve context dependency. The methods introduced in the Sections 4.3 and 4.4 are based on the histogram difference operation which is re-defined in this work.

2.3 Research Problem

There are a number of existing methods for approximate string matching, as described in Section 2.4. Nevertheless, the number of knowledge domains applying ASM methods has grown since the 1960s, as shown in Sections 1.1.1 to 1.1.8.

The life and social sciences, bioinformatics and medical informatics have experienced a large growth in the amount of electronic data produced by a number of sources. To reduce the labor costs of data mining and processing applications, the ASM methods are widely employed in the above-mentioned domains.

The author hypothesizes that the performance of existing, widely used similarity metrics may be improved for duplicate detection and clustering tasks. The

author will show empirically that this hypothesis holds true in Section 5.2 through Section 5.7 by presenting the evaluation results of various string similarity metrics. Based on these results, the author claims that there is a scientific need for new string similarity metrics capable of improving performance in certain domains.

The author presents the previously introduced and novel string similarity algorithms. An extensive evaluation is performed for duplicate detection and clustering tasks. The developed methods are benchmarked against ten well-known similarity functions. The proposed methods achieve superior results measured in average precision, maximum F_1, and execution time.

2.3.1 Research Problem in Life and Social Sciences Domain

In the life and social sciences domain, ASM methods are used for creation, maintenance, and duplicate identification and removal in name and address lists. It is known that many datasets have typographical errors in more than 20% of first names and also in last names. The application of ASM methods significantly improves matching efficiency and facilitates the labor-intensive manual matching process [89].

In the life and social sciences domain, as well as in other areas, the problem of duplicate detection arises in the applications which collect and extract data from Web pages or other unstructured or semi-structured documents. In this case, ASM methods are applied at the data cleaning step, which occurs before uploading records into a database [90]. An example of a business-related information integration application that uses data from the Web is TheaterLoc [91], which collects and processes restaurant and movie theater data from the Internet, locates objects on a map and allows site visitors to view restaurant reviews, movie show times and trailers.

2.3.2 Research Problem in Bioinformatics Domain

A vast number of biological entities such as genes and proteins are available in the biological databases Swiss-Prot [1], GenBank [2], and others. These databases serve as critical resources for molecular biologists to conduct their research. Over the last two decades, the rapid development of the biological databases has been driven by an explosive growth of data due to the high throughput sequencing and automation in genomics and proteomics. For example, the most recent statistical report of SwissProt shows that the number of the SwissProt sequences has grown six times over four years from two million sequences in 2006 to twelve million sequences in 2010.

Table 2.1 Sample Duplicate Entities

Field	Swiss-Prot Record	PIR Record
Locus ID	P34180	S22388
Definition	Phospholipase A2, neutral precursor (Ammodytin I2) (Phosphatidylcholine 2-acylhydrolase).	phospholipase A2 (EC 3.1.1.4) ammodytin I2 precursor - western sand viper.
Database source	Swiss-Prot: locus PA2N_VIPAA, accession P34180;	PIR: locus S22388
Organism	Vipera ammodytes ammodytes	Vipera ammodytes ammodytes
Sequence	MRTLWIVAVCLIGVE GNLYQFGNMIFKMTK KSALLSYSNYGCYCG WGGKGKPQDATDRC CFVHDCCYGRVNGC DPKLSIYSYSFENGDI VCGGDDPCLRAVCEC DRVAAICFGENLNTY DKKYKNYPSSHCTET EQC	MRTLWIVAVCLIGVE GNLYQFGNMIFKMTK KSALLSYSNYGCYCG WGGKGKPQDATDRC CFVHDCCYGRVNGC DPKLSIYSYSFENGDI VCGGDDPCLRAVCEC DRVAAICFGENLNTY DKKYKNYPSSHCTET EQC

Even though research efforts, such as the UniProt Knowledgebase, work on integration of many biological databases, progress is still far from satisfactory. The

most common sources of errors contributing to the low quality of public sequence databases are as follows [3]:

1. Lack of cross-referencing. The same sequence may be entered into more than one database without cross-referencing these records.

2. Duplicate entries. The sequence is submitted more than once to the same database.

3. Duplicate annotations. The annotations of the same sequence are submitted separately by different research groups.

Table 2.1 shows the same protein found in PIR and Swiss-Prot without cross-referencing between the two records [4]. This example demonstrates the high degree of string similarity between definitions of the duplicate entries. This factor justifies the use of string comparison techniques to deal with such typographical variations.

As biological databases become more pervasive, various data quality concerns are emerging. The aforementioned quality issues are non-trivial and can cause many problems for the database. For the biological data to be corrected and standardized, methods and frameworks must be developed to handle both structural and traditional data.

In the field of information integration, duplicate detection has been widely studied [5]. Among various techniques proposed in the field, string similarity provides an unsupervised statistical model and has been applied in many different applications [6, 7]. String similarity and matching algorithms are used for entity matching by measuring individual record fields, since individual fields are often stored as strings. The widely-used notion of string similarity is the Levenshtein edit distance: the minimum number of insertions, deletions, and substitutions required to transform one string into another [8]. Advantages gained by entity matching based on string

similarity are removing duplicate biological data [9], discovering substructures in biochemical molecules [10], and detecting duplicate 2D Nuclear Magnetic Resonance (NMR) spectra for the structural analysis of molecules [11].

The performance of numerous string similarity metrics varies from dataset to dataset due to the structure of the data and other textual characteristics, such as length of the matched strings, word frequency, logical organization of strings, etc. This phenomenon was empirically shown, as described in Chapter 5 of this work. As the authors' experiments agree with research by Tan et al. [125], a single universal string metric with superior performance on various types of datasets does not exist. Still, efforts to create better methods continue.

One direction of ongoing research involves accounting for the contextual dependencies in texts for performance improvement. Several major research efforts based on a probabilistic approach are introduced in Section 2.2 of this work. Another significant effort in this field was made by Wei [51]. Wei introduced the Markov Edit Distance (MED), which was one of the initial applications of the Markov model to the string metric domain. Wei [51] suggested that the MRF, with its solid theoretical mathematical background and practical success in many disciplines, was likely to shed more light on how to build a salient framework that would render the edit distance concept more powerful. Wei presented two modifications of the string metric based on Markov's ideas: the reshuffling MED capable of handling reshuffling relations among patterns; and the coherence MED, which allows more complex operations on sub-patterns such as insertion, deletion, and substitution based on local contextual dependencies.

2.3.3 Research Problem in Medical Informatics Domain

Launched in 1986, the Unified Medical Language System (UMLS) remains a well-known long-term research effort [79] to develop an extensive terminological knowledge base consisting of three major components: the Metathesaurus, the Semantic Network, and the SPECIALIST Lexicon. The UMLS 2011AA release contains more than 2.4 million concepts and almost 10 million unique terms, retrieved from 160 source vocabularies [93].

Source integration is a complicated multi-step process demanding a vigorous research effort. Although many algorithmic aides are available to support experts who are adding to, updating or auditing vocabularies, still no solution has been found to solve these problems without extensive human interaction. It is planned to integrate even more sources into the UMLS in the future [79]. Furthermore, new versions of existing vocabularies require reintegration into the UMLS as a part of its update cycle. Therefore, developing new techniques and improving existing ones for term matching for the UMLS remain important tasks.

In Section 4.4, the author proposes the Longest Approximately Common Prefix (LACP) method as a context-sensitive algorithm for improving existing source integration and auditing techniques. The LACP could be included as one data processing step into existing text-to-thesaurus mapping programs such as CLARIT, SAPHIRE, Metaphrase, MetaMap/MMTx [94], MicroMeSH or integration techniques such as Piecewise Synonym Generation [79] in order to improve the precision of the results. It is worth noting that the LACP does not perform the kinds of text manipulations that the well-known SPECIALIST lexicon tools Norm, Word Index, or LVG [95] do, but it assesses the similarity or dissimilarity of two strings.

2.4 String Distances

2.4.1 Edit Distance

In order to define string metrics, many researchers use the notation introduced in the classic paper of Wagner and Fisher [21] and adopted by others [25]. Let Σ be a finite alphabet and Σ^* be the set of all finite length strings over Σ. Let $S = S_1S_2 \ldots S_n$ be a string of Σ^*, where S_i is the i-th symbol of S. Let $S_{i...j}$ be the substring of S consisting of consecutive symbols from S_i to S_j where $1 \leq i \leq n$, $1 \leq j \leq n$. The length of the string S is $|S| = n$ and the length of the substring $|S_{i...j}| = j - i + 1$. When $i > j$, $S_{i...j}$ is the null string λ, $| \lambda | = 0$. $|S|$ denotes the length of the string S defined as the number of characters in a string.

A simple edit operation is a pair $(a, b) \neq \lambda$ of strings where a and b are strings of length 1 or 0. The notation $a \rightarrow b$ is used for an edit operation (a, b). There are three commonly used edit operations:

- The insertion: $a \rightarrow b$, $a \neq \lambda$, $b = \lambda$;
- The deletion: $a \rightarrow b$, $a = \lambda$, $b \neq \lambda$;
- The substitution: $a \rightarrow b$, $a \neq \lambda$, $b \neq \lambda$.

The edit transformation of string S into string T is the sequence E of the elementary edit operations that transform S into T. The elementary edit operations are assigned weights by the weight function ω, which assigns to each edit operation (a, b) a real number $\omega(a, b) = r \geq 0$. The weight function ω can be extended to a sequence E as follows: $\omega(E) = \sum_{i=1}^{m} \omega(E_i)$, where m is the length of E.

For the strings S and $T \in \Sigma^*$, the edit distance between S and T is defined as follows:

$$\delta(S, T) = \min \{ \omega(E) \mid E \text{ is the edit transformation of } S \text{ into } T \} \qquad (2.1)$$

In the classic case, the assumption is that $\omega(a, b) = \delta(a, b)$ with the following constraints: $\omega(a,a)=0$ and $\omega(a, b) + \omega(b, c) \geq \omega(a, c)$, known as the triangle inequality. Furthermore, by adding the assumption that δ is symmetric, i.e. $\delta(a, b) = \delta(b, a)$, and strictly positive on each edit operation, i.e. $\delta(a, b) > 0$ where $a \neq b$, the classic approach interprets δ as a metric on a space of all strings, thereby explaining the term "distance."

2.4.2 Traces

The trace $R_{S,T}$ from a string S to a string T is the sequence of the ordered pairs of integers (i, j) satisfying the following conditions:

- $1 \leq i \leq n$, $1 \leq j \leq m$ where $|S| = n$ and $|T| = m$;
- For any two distinct pairs (i_1, j_1) and $(i_2\ j_2)$ in the trace R the following conditions hold:
    a. $i_1 \neq i_2$ and $j_1 \neq j_2$;
    b. $i_1 < i_2$ if and only if $j_1 < j_2$.

Traces denote possible paths from the pair of initial characters of S and T to any other pair of characters from S and T. The sequence forming the trace $R_{S,T}$ may correspond to a complete or incomplete transformation from S to T.

2.4.3 Edit Path

The edit path $P_{S,T}$ between two strings S and T is the sequence of ordered pairs of integers (i_k, j_k) where $0 \leq k \leq m$ such that:

1. $0 \leq i_k \leq |S|$;
2. $0 \leq j_k \leq |T|$;
3. $(i_0, j_0) = (0, 0)$;
4. $(i_k, j_k) = (|S|, |T|)$;
5. $0 \leq i_k - i_{k-1} \leq 1$, $\forall k \geq 1$;
6. $0 \leq j_k - j_{k-1} \leq 1$, $\forall k \geq 1$;
7. $i_k - i_{k-1} + j_k - j_{k-1} \geq 1$.

Conditions (1) and (2) specify the ranges for i_k and j_k. Conditions (3) and (4) ensure that the traversal starts at the pair of initial characters of S and T and ends with the pair of final characters of S and T. Conditions (5) and (6) ensure that during a single step no more than one character is traversed in the horizontal, vertical, or diagonal direction. Condition (7) assures that at the k-th step a horizontal, vertical, or diagonal move is actually made.

Each pair of successive points of the edit path corresponds to one edit operation of insertion, deletion, or substitution. By traversing the edit path $P_{S,T}$ from the beginning to the very end, the full transformation of the string S into the string T can be restored.

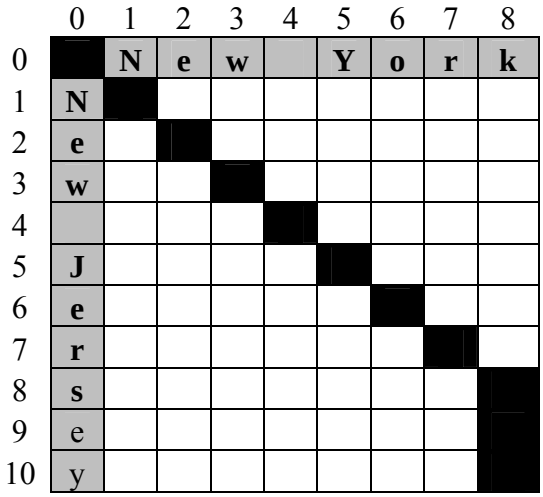


Figure 2.1 An example of the edit path between two strings.

Figure 2.1 shows a graphical representation of the edit path between the strings "New York" and "New Jersey." The black cells correspond to the steps of the edit path. Here, the edit path $P_{S,T}$ = {(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (8, 9), (8, 10)}.

The weight of the edit path can be computed by the following formula:

$$W(P_{S,T}) = \sum_{k=1}^{m} \omega(S_{i_k}, T_{i_k}) \qquad (2.2)$$

Now, the formula for the edit distance takes this form:

$$\delta(S, T) = \min \{ W(P) \mid P \text{ is the edit path between } S \text{ and } T\} \quad (2.3)$$

The recursive formula below for the Levenshtein edit distance was introduced by Wagner and Fisher [21]:

$$\begin{aligned}\delta(S_{1...i}, T_{1...j}) = \min \{ & \delta(S_{1...i-1}, T_{1...j}) + \omega(S_i, \lambda), \\ & \delta(S_{1...i-1}, T_{1...j-1}) + \omega(S_i, T_j), \\ & \delta(S_{1...i}, T_{1...j-1}) + \omega(\lambda, T_j)\}\end{aligned} \quad (2.4)$$

In (2.4), $S_{1...i}$ denotes the substring of the string S, where $S_{1...i} = S_1S_2...S_i$ and $T_{1...j}$ denotes the substring of the string T, where $T_{1...j} = T_1T_2...T_j$. The application of the formula (2.4) to the transformation of the string "New York" into "New Jersey" is shown below in Figure 2.2. Each inner cell with a white background contains a value of the edit distance for the substrings $S_{1...k}$ and $T_{1...l}$ where k is the x-coordinate of the cell and l is its y-coordinate. The final value of the Levenshtein edit distance is shown in the bottom right cell with the coordinates (8, 10) and the value 5.

	0	1	2	3	4	5	6	7	8
0		**N**	**e**	**w**		**Y**	**o**	**r**	**k**
1	**N**	0	1	2	3	4	5	6	7
2	**e**	1	0	1	2	3	4	5	6
3	**w**	2	1	0	1	2	3	4	5
4		3	2	1	0	1	2	3	4
5	**J**	4	3	2	1	1	2	3	4
6	**e**	5	4	3	2	2	2	3	4
7	**r**	6	5	4	3	3	3	2	3
8	**s**	7	6	5	4	4	4	3	3
9	**e**	8	7	6	5	5	5	4	4
10	**y**	9	8	7	6	5	6	5	5

Figure 2.2 The Levenshtein distance matrix for S = "New York", T = "New Jersey."

2.4.4 Post-Normalized Edit Distance

The Post-Normalized Edit Distance (PNED) is a simple approach to take into account the lengths of strings for which the metric is being computed. For this variation of the algorithm, the value of the edit distance computed using the Levenshtein function is

divided by the length of the edit path. As an example, consider a pair of strings of length two differing by a single character and a pair of strings of length one hundred with one non-matching character. In the first case, the dissimilarity is 50%, but in the second case it is only 1%. To stress this difference, the post-normalization is applied as follows:

$$PNED(S,T) = \frac{\delta(S,T)}{|P_{S,T}|} \tag{2.5}$$

where $|P_{S,T}|$ is the length of edit path $P_{S,T}$.

2.4.5 Normalized Edit Distance

The Normalized Edit Distance (NED) [25] differs from the PNED by the way in which the normalization is done. In case of the NED, the final value is computed within a single dynamic programming process by minimizing the normalized weight of the edit path P:

$$\hat{W}(P_{S,T}) = \frac{W(P_{S,T})}{|P_{S,T}|} \tag{2.6}$$

Then the expression for the NED takes the form:

$$NED(S,T) = \min\{\hat{W}(P_{S,T})\} \tag{2.7}$$

Marzal and Vidal [25] showed that the NED outperforms the PNED significantly. Also the researchers proved that the minimization (2.7) cannot be substituted by first minimizing $W(P_{S,T})$ and then normalizing it by the length of the obtained edit path $P_{S,T}$.

2.4.6 Damerau-Levenshtein Distance

The Damerau-Levenshtein Distance (DLD) [33] expands the set of allowed edit operations by adding a transposition, which does not satisfy the definition of the simple edit operations. The transposition is defined as $ab \rightarrow ba$, $a \neq \lambda$, $b \neq \lambda$, $a \neq b$.

The formula for the DLD is

$$DLD = \begin{cases} \delta(S_{i-2}, T_{j-2}) + \omega(S_{i-1}S_i, S_iS_{i-1}) & when \quad S_{i-1} = T_j, S_i = T_{j-1} \\ \delta(S_{1\ldots i}, T_{1\ldots j}) & otherwise \end{cases} \tag{2.8}$$

2.4.7 Jaro Metric

The Jaro metric [26] is not based on the edit distance model. Its positive results in the record linkage domain are due to the consideration of the number and order of the common characters in two strings. A character S_i from a string S is defined to be common with a string T when $S_i = T_j$ and $i - H \leq j \leq i + H$, where

$$H = \frac{\min(|S|, |T|)}{2} \tag{2.9}$$

As an example, consider strings S = "aaabaa" and T = "ccbccccc". Here, i=4, j=3, character S_4 is equal to character T_3, and H = min(6, 8)/2 = 3. Now, both conditions for character "b" are satisfied: $S_4 = T_3$ and $i - 3 \leq j \leq i + 3$. Thus, S_4 is common with T_3.

Let $S' = S_1 \ldots S_K$ be the characters in S, which are common with T and appear in the same order as they originally appear in S. Let $T' = T_1 \ldots T_k$ be the characters in T, which are common with S and appear in the same order as they originally appear in T. The transposition for S' and T' is the position i such that $S'_i \neq T'_i$. Let $J_{S',T'}$ be half the number of the transpositions for S' and T' [28]. The Jaro similarity metric is expressed as follows:

$$Jaro(S,T) = \frac{1}{3} \cdot \left(\frac{|S'|}{|S|} + \frac{|T'|}{|T|} + \frac{|S'| - J_{S',T'}}{|S'|} \right) \tag{2.10}$$

2.4.8 Jaro-Winkler Metric

Winkler's modification of the Jaro metric [34] shows a significant improvement in the experimental results [35]. This algorithm recalculates the score of the Jaro metric based on the length of the common prefix. Let *Pref* be the longest common prefix shared by the strings *S* and *T*. Let *P'*= max(*Pref*, 4). Then the Jaro-Winkler metric is defined as

$$Jaro-Winkler(S,T) = Jaro(S,T) + \frac{P'}{10} \cdot (1 - Jaro(S,T)) \tag{2.11}$$

2.4.9 Jaccard Similarity

The Jaccard similarity [36] is the token-based distance metric, which considers the strings *S* and *T* as sets of tokens. This method interprets a token as a single word. It was first introduced in research on Alpine flora diversity in 1912. The Jaccard similarity is

$$Jaccard = \frac{|X \cap Y|}{|X \cup Y|} \tag{2.12}$$

where *X* is the set of tokens of the string *S*, and *Y* is the set of tokens of the string *T*.

2.4.10 Smith-Waterman Algorithm

This algorithm is considered to be the local version of the dynamic programming algorithm for sequence alignment [37]. It is designed to find the best alignment between the subsequences of two long sequences in the bioinformatics domain. The Smith-Waterman algorithm [38] assigns 0 to the complete mismatch and the highest score to the "best local alignment" [37].

$$SW(S_i, T_j) = \begin{cases} 0, \text{ when } 0 \le i \le |S| \text{ or } 0 \le i \le |T| \\ \max \begin{cases} 0 \\ SW(S_{i-1}, T_{j-1}) + \omega_{match}(S_i, T_j) & S_i = T_j \\ SW(S_{i-1}, T_{j-1}) + \omega_{mismatch}(S_i, T_j) & S_i \neq T_j \\ SW(S_{i-1}, T_j) + \omega_{deletion}(S_i, \lambda) & S_i \neq T_j, T_j = \lambda \\ SW(S_{i-1}, T_j) + \omega_{insertion}(\lambda, T_j) & S_i \neq T_j, S_j = \lambda \end{cases} \end{cases} \quad (2.13)$$

This algorithm assigns different weights depending on the match, mismatch, deletion or insertion of a pair of characters. This feature also makes it possible to attribute the variable costs to a gap when aligning two strings. In this case, the gap costs are $\omega_{deletion}(S_i, \lambda)$ and $\omega_{insertion}(\lambda, T_j)$.

2.4.11 Gotoh Algorithm

An extension of the Smith-Waterman technique was introduced by Gotoh [39] by adding the affine gap penalty. The main idea of this method is to impose a penalty for opening a gap and another smaller penalty for extending the gap. The Gotoh algorithm uses three matrices: *G* for storing the best scores for the alignment S_i and T_j which ends without a gap; *H* for the best scores with the condition that the alignment of $S_{1..i}$ and $T_{1..j}$ ends with a gap in *S*; and *V* for the best scores with the condition that the alignment ends with a gap in *T*. The Gotoh algorithm is defined as follows:

$$\begin{aligned} &G(S_0, T_0) = 0 \\ &G(S_0, T_j) = H(S_0, T_j) = g(j), \quad 1 \le j \le |T| \\ &G(S_i, T_0) = V(S_i, T_0) = g(j), \quad 1 \le i \le |S| \\ &H(S_i, T_0) = -\infty, \quad 0 \le i \le |S| \\ &V(S_0, T_j) = -\infty, \quad 0 \le j \le |T| \\ &G(S_i, T_j) = \max \begin{cases} G(S_{i-1}, T_{j-1}) + \omega(S_i, T_j) \\ H(S_i, T_j) \\ V(S_i, T_j) \end{cases}, \quad \begin{matrix} 1 \le i \le |S| \\ 1 \le j \le |T| \end{matrix} \end{aligned} \quad (2.14)$$

$$H(S_i,T_j)=\max\begin{cases}G(S_i,T_{j-1})+gapstart+gapextend\\ H(S_i,T_{j-1})+gapextend\end{cases},\quad \begin{matrix}1\le i\le|S|\\ 1\le j\le|T|\end{matrix}$$

$$H(S_i,T_j)=\max\begin{cases}G(S_{i-1},T_j)+gapstart+gapextend\\ V(S_{i-1},T_j)+gapextend\end{cases},\quad \begin{matrix}1\le i\le|S|\\ 1\le j\le|T|\end{matrix}$$

where ω is a weight function, g is the affine gap cost function, *gapstart* is a cost to start a gap, *gapextend* is a cost to extend a gap. According to Gotoh [39], weights and parameters cannot be determined a priori, but they may be estimated by a dynamic optimization procedure.

2.4.12 Monge-Elkan Algorithm

The Monge-Elkan similarity function [40] makes use of the affine gaps by implementing the Gotoh metric to match fields of a record (or words of a string in string distance terminology). The Gotoh distance is used to find the most similar word in T for each word in S. Then these scores are combined in order to get the final similarity value using the following formula:

$$ME(S,T)=\frac{1}{|S|}\sum_{i=1}^{|S|}\max_{j=1}^{|T|}(Gotoh(S_i,T_j)) \tag{2.15}$$

where *Gotoh*() is the Gotoh function.

2.4.13 Needleman-Wunsch Algorithm

Unlike the Smith-Waterman algorithm [38], the Needleman-Wunsch method [22] looks for the best global alignment. This method is also known as the Sellers algorithm [66]. It is similar to the Levenshtein distance [20] but with parameterized values for the insertion and deletion operations. When the gap cost g in the

Needleman-Wunsch algorithm is set to 1, it becomes an exact copy of the Levenshtein metric.

$$NW(S_i,T_j)=\max\begin{cases}NW(S_{i-1},T_{j-1})+\omega(S_i,T_j)\\ NW(S_{i-1},T_j)+g\\ NW(S_i,T_{j-1})+g\end{cases} \tag{2.16}$$

where g is the gap cost.

2.4.14 TFIDF Algorithm

Term Frequency-Inverse Document Frequency (TFIDF), also known as the Cosine Similarity [41], is a widely used token-based algorithm in information retrieval [43, 44]. This method calculates a term relevance weight defined as the proportion of relevant documents in which the term occurs divided by the proportion of non-relevant items in which the term occurs [42]. As defined by Cohen et al. [28]:

$$\begin{aligned}TFIDF(S,T)&=\sum_{\omega\in S\cap T}V(\omega,S)\cdot V(\omega,T)\\ V(\omega,S)&=\frac{V'(\omega,S)}{\sqrt{\sum_{\omega'}V'(\omega,S)^2}}\\ V'(\omega,S)&=\log(TF_{\omega,S}+1)\cdot\log(IDF_{\omega})\end{aligned} \tag{2.17}$$

where $TF_{\omega,S}$ is the frequency of the word ω in S, N is the size of the text, and IDF_{ω} is the inverse of the fraction of terms in the corpus that contains ω.

2.4.15 Soft TFIDF

The Soft TFIDF method was proposed by Cohen et al. [28] as a hybrid distance function combining TFIDF with a secondary similarity function. The Monge-Elkan [40], Jaro [26], and Jaro-Winkler [27] methods were used by Cohen et al. in the experimental phase.

$$SoftTFIDF(S,T) = \sum_{\omega \in CLOSE(\theta,S,T)} V(\omega,S) \cdot V(\omega,T) \cdot D(\omega,T) \quad (2.18)$$

where $CLOSE(\theta, S, T)$ is the set of words $\omega \in S$ such that $\exists \upsilon \in T$ and $dist'(\omega,\upsilon) > \theta$; for $\omega \in CLOSE(\theta, S, T)$, $D(\omega,T) = \max_{\upsilon \in T} \text{dist}(\omega,\upsilon)$. In other words, Soft TFIDF does not discard words which match approximately, unlike TFIDF which keeps only exact matches.

2.4.16 Information Distance

Information Distance [121] is considered by Vitanyi et al. [122] to be a universal distance measure for objects of all kinds. Unfortunately, Information Distance, as defined by him, is incomputable, since it is based on the Kolmogorov complexity. Nevertheless, Vitanyi et al. [122] suggest two approximations of this method. The first approximation lies in the calculation of the difference in the number of bytes between two compressed files. It is intended for objects which could be represented by strings. The second version uses World Wide Web search engine results to target names and abstract concepts. In [122], Information Distance is applied to clustering and other tasks.

The Information Distance method was selected for benchmarking, since its authors claim its universality, and it has shown promising results in clustering tasks [122]. For this work, approximation via compression was selected. Bzip2, PPMZ, and Gzip are mentioned as appropriate implementations of compression methods [122]. In this work, the Gzip algorithm was selected, since it is available in the Java Software Development Kit used for implementation of the other evaluated methods.

The formula representing the Information Distance is shown below [121]:

$$d_{\min}(x,y) = \frac{\min\{(K_U(x \mid y), K_U(y \mid x)\}}{\min\{K_U(x), K_U(y)\}} \tag{2.19}$$

where $d_{\min}(x, y)$ is the Information Distance and K_U is the Kolmogorov complexity for the universal Turing machine. The definition of the Kolmogorov complexity is defined as follows [121]:

$$K_U(x \mid y) = K_{U'}(x,y) + C \tag{2.20}$$

where $K_U(x, y)$ is the Kolmogorov complexity of a binary string x conditional on another binary string y, given a universal Turing machine U. U' is a different Turing machine; the constant C depends only on U'.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Data Sources

3.1.1 Life and Social Sciences Data Sources

The five datasets depicted in Table 3.1 were selected to measure the performance of string metrics in text retrieval applications. In the Animals and Birds datasets, common names are used as a primary key and scientific names as a secondary key. In the Restaurants dataset, the manually constructed secondary keys correspond to the real-world restaurant data consisting of a name, address, phone number, and a brief description of the cuisine served. The Parks dataset was built by Cohen et al. [45] from the online park directories linking the park names to the URLs of the corresponding sites. In this dataset, park names are used as a primary key and the URLs as a secondary key. The Census dataset contains census-like data: the database record ID, last name, first name, middle initial, street number and name.

The datasets described above contain duplicate records in the sense that the same dataset has more than one record corresponding to the same entity. The primary keys are the exact entity identifiers. The secondary keys are the non-matching string fields containing entity data.

Table 3.1 Datasets used for the string metrics evaluation

Dataset	# of Records	Source
Animals	5,709	[45]
Birds	982	[45]
Census	841	[28]
Parks	654	[45]
Restaurants	863	[46]

Table 3.2 shows the extraction of duplicate records from the Census dataset in order to identify households living at the same address. The string similarity metric is applied to pairs of secondary key values to determine whether a pair of records describes members of the same household. Then the correctness of the decision is checked by comparing the primary key values: match means the decision is correct and non-match that it is incorrect.

Table 3.2 Duplicate records retrieved from the Census dataset

Primary Key	Secondary Key
ID445012723835840000	COBY LASHIWN Y 303 MAIN
ID445012723835840000	COBY WILIAMS A 303 MAIN
ID445012723835840000	COBY ANGELA 303 MAIN
ID445012723835840000	COBY MIKE D 303 MAIN
ID445012723837740000	REEVES DOUGLASS F 625 MARTIN LUTHER K
ID445012723837740000	REEVES NWAMAKA M 625 MARTIN LUTHER K
ID445012723837740000	REEVES WILLY L 625 MARTIN LUTHER K
ID445012723838570000	HEAVAENER FLORRE 608 OCONEE
ID445012723838570000	HEAVAENER WILSREVO 608 OCONEE
ID445012723838570000	HEAVAENER JEFFREY S 608 OCONEE
ID445012723840870000	SOLLIVAN ANNE G 14245 22
ID445012723840870000	SOLLIVAN EVENS 14245 22
ID445012723840870000	SOLLIVAN BRANDIE D 14245 22

3.1.2 Bioinformatics Data Sources

The author uses the following UniProt GOA Proteome Sets to compare the performance of the Markov Random Field Edit Distance (or MRFED), the normalized edit distance, to other algorithms for the entity matching tasks as shown in Table 3.3.

The Gene Ontology Annotation (GOA) database was developed to provide high-quality supplementary Gene Ontology (GO) annotations for proteins in the UniProt Knowledgebase [57]. For duplicate identification, the correct answer datasets were constructed from the UniProt GOA Proteome [58]. The GOA Proteome data provide researchers with an extensive testing ground for a duplication detection task.

Table 3.3 Bioinformatics datasets used in experiments

Organism	Entries Annotated	GO Annotations
Paramecium Tetraurelia	217	1,402
Bacteriophage T4	116	623
Carsonella Ruddii	141	1,336
Hyperthermus Butylicus	918	5,625
Buchnera Aphidicola Cedri Cinara	339	4,140

Each GOA contains 15 attributes described in Table 3.4. The author used all features except the database code and the unique ID from the GOA to compute the inter-similarity among entries, i.e., the author measured the metadata identity to find similarities. The unique IDs were used to check the correctness of a match/non-match decision. The GOA uses the International Protein Index, where the sequence identifiers from the GOA, Ensembl, H-Invitational Database, TAIR, RefSeq and Vega groups are combined to provide the species-specific annotation sets.

Table 3.4 Attribute descriptions of GOA

Column	Description
DB	Database from which an annotated entry has been taken
DB_Object_ID	Unique identifier
DB_Object_Symbol	Symbol (unique and valid)
Qualifier	Flag that modifies the interpretation of the annotation
GO ID	GO identifier
DB:Reference	Reference cited to support the annotation
Evidence	Evidence for the annotation
With	Additional identifier
Aspect	One of three ontologies
DB_Object_Name	Name of the gene or gene product
Synonym	Gene symbol
DB_Object_Type	Entity annotated
Taxon ID	Identifier for the species
Date	Date of the last annotation
Assigned By	Source of the annotation

The biological records contain the following three main field types [4]:

- The sequences themselves, e.g. protein and DNA sequences;
- The categorical fields;
- The free-text strings.

In order to detect the duplicates, the authors' approach measures the categorical and free-text strings.

3.1.3 Medical Informatics Data Sources

Medical informatics datasets were obtained from the 2009AB version of the UMLS. As the UMLS contains terms from many sources, the author produced two types of datasets:

- Records selected from the multiple UMLS sources;
- Records selected from the SNOMED CT subset.

There are several reasons to choose these datasets. The multiple-source datasets make it possible to perform overall evaluations in medical informatics. The UMLS contains biomedical terms from many sources, allows for the integration of new sources, and permits researchers to continually audit existing sources over time. By performing experiments on a "multiple source dataset," the author addresses, to some degree, the problem of how to integrate a new source into the UMLS. The SNOMED CT was selected for evaluation because of its wide use and practical importance [92].

The datasets were built by applying custom-built SQL queries to a MySQL database using the Metathesaurus. The database was populated using the scripts provided with the 2009AB UMLS distribution. Analogously to the life and social sciences datasets described in Section 3.1.1, concept unique identifiers (CUI) were used as a primary key, and string representations of the terms were employed as a secondary key. Two or more records with the same primary key correspond to the same concept. Such records are considered as duplicates in the experimental phase.

Table 3.5 depicts characteristics of the biomedical datasets, including the numbers of unique terms and concepts. There is a notable difference between the numbers of unique concepts in datasets (2) and (4). It is possible to explain this as follows: dataset (2) contains records from the same terminology and thus has a higher chance of retrieving terms describing the same concept. In contrast, dataset (4) consists of records collected from multiple sources, which decreases this probability.

Table 3.5 Medical informatics datasets used in experiments

#	Dataset	# of Concepts	# of Terms
1	SNOMED-most frequent concepts	155	5,000
2	SNOMED-longest concepts	1,805	5,000
3	UMLS-most frequent concepts from multiple sources	100	4,979
4	UMLS-longest concepts	3,337	5,000

Table 3.6 Duplicate records from the "SNOMED-most frequent concepts" dataset

Primary Key	Secondary Key
C0034606	Diagn. nuclear medicine NOS
C0034606	Diagn. nuclear medicine NOS (procedure)
C0034606	Diagnostic nuclear med.
C0034606	Diagnostic nuclear medicine
C0034606	Diagnostic nuclear medicine NOS
C0034606	Diagnostic nuclear medicine NOS (procedure)
C0034606	Diagnostic radionuclide study
C0034606	Diagnostic radionuclide study, NOS
C0034606	NM - Nuclear medicine
C0034606	Nuclear med.-diagnostic
C0034606	Nuclear medicine
C0034606	Nuclear medicine diagnostic procedure

Table 3.6 shows duplicate records retrieved from dataset (1). The most frequent concept datasets (1) and (3) were derived by getting the top records out of the record sets sorted in descending order by the number of non-matching terms belonging to the same concept. The longest concept datasets (2) and (4) were obtained

by selecting the top records out of the record sets sorted in descending order by the length of the term strings.

3.2 Details of the Methodology

The standard approach to evaluating record linkage or duplicate detection systems employs the notions of relevant and non-relevant records. For the task of duplicate detection, a record is given a designation as a duplicate or a mismatch compared to another record. When a decision about duplicate records is made correctly, a relevant record is retrieved; when a wrong decision is made, a non-relevant record is recovered [47].

For the life and social sciences datasets presented in Section 3.1.1, matching is performed on the secondary keys such as names, addresses and phone numbers. Then this decision is assessed in terms of relevance to the primary key. If two records are identified as duplicates by the secondary keys and have the same primary key, this decision is considered relevant. Otherwise, when two records are identified by the system as duplicates but have non-matching primary keys, the decision is non-relevant.

The author uses the conventional approach of information retrieval to evaluate the performance of string distance metrics [47]. The two most frequent measures for information retrieval effectiveness are precision and recall (see formulas (3.3) and (3.4)). Evaluation based on these two measures concentrates on true positives by examining the quantities of returned relevant documents with respect to the number of false positives. The term *documents* is used here in a broad sense and may refer to the document elements, sentences of phrases, or textual records apart from regular documents.

It is important to calculate both precision and recall because gaining higher precision is preferable for certain tasks while higher recall is more valuable for other information retrieval problems. In some cases, high precision results are of primordial importance, when the extraction results are not manually controlled, while in other cases, where the machine extraction is only performing an initial filtering of the information that eventually is manually selected, a high recall of the extraction is more important [83].

The high value of precision obtained in some experiments means that the retrieved data either have a small number of errors or no errors at all. The high recall values correspond to those experiments in which all or almost all of the information that needs to be extracted is actually extracted [83].

Precision and recall are tradeoffs against one another: on the one hand, it is possible to obtain the maximum value of recall with a low value of precision by retrieving all documents for all queries. On the other hand, the precision usually decreases as the number of retrieved documents grows. A single measure that trades off precision versus recall is the *F* measure, which is the weighted harmonic mean of precision and recall [48]. The *F* measure, derived from the *E* measure, is the most commonly used metric for combining precision and recall into one metric.

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{3.1}$$

$$\beta^2 = \frac{1 - \alpha}{\alpha} \tag{3.2}$$

where $\alpha \in [0, 1]$ and $\beta^2 \in [0, \infty)$. The default balanced *F* measure equally weights precision and recall. It is achieved when $\alpha = 0.5$ or $\beta = 1$. It is commonly written as F_1, which is short for $F_{\beta=1}$ [48].

For the evaluation of the string distance performance, the author calculated the average precision and maximum F_1 (see formula (3.5)), and built the precision-recall curves for several string similarity techniques evaluated on the datasets mentioned in Section 3.1.

The precision-recall curves became a widespread conceptual tool for assessing classification performance. The curves relate the precision of a classifier to its true positive rate. The precision-recall curves offer a scale-adapted graphical display that makes it possible to visualize and rank performance more easily when the theoretical proportion of the positive instances is small compared to the total number of records [84].

Second String [28], an open-source Java toolkit, was used as the experimental test bed. In experiments, each term was matched against all other terms within a set of candidate pairs from the same dataset. The goal was to determine whether every pair of terms had the same identifier.

$$P = \frac{D_r}{D_t} \tag{3.3}$$

$$R = \frac{D_r}{N_r} \tag{3.4}$$

$$F_1 = \frac{2P \cdot R}{P + R} \tag{3.5}$$

Formulae (3.3) - (3.5) use the following notation: precision P, recall R, harmonic mean F_1, the number of relevant items retrieved D_r, the number of relevant items in a collection N_r, and the total number of retrieved items D_t. In (3.5), the number 2 in the numerator indicates that recall and precision are of equal importance.

The precision-recall curves consist of the interpolated precision taken at eleven recall points 0, 0.1, ..., 1. The interpolated precision P_{interp} at recall level R is defined as the highest precision found for any recall level $R' \geq R$ [48]:

$$P_{\text{interp}} = \max_{R' \geq R} P(R') \tag{3.6}$$

CHAPTER 4

SIMILARITY FUNCTIONS FOR DUPLICATE DETECTION AND CLUSTERING TASKS

4.1 Markov Random Field Edit Distance

4.1.1 Background

In this section, several terms related to the Markov Random Field theory are described. The MRFs incorporate spatial and contextual dependencies by means of neighborhood systems and cliques. The author implements MRF theory using a string distance method to improve matching accuracy in comparison to edit distances constructed with symbol-based cost functions. A Neighborhood System (NS) consists of a set of nodes *S*. Each node is a pair of characters (s_i, t_j), where s_i is the *i*-th character of the first string participating in the alignment, and t_j is the *j*-th character of the second string. Li [52] defines the Neighborhood System *N* as

$$N = \{N_i \mid \forall i \in S\} \tag{4.1}$$

and

$$N_i = \{i' \in S \mid [dist(p_{i'}, p_i)]^2 \leq d, i' \neq i\} \tag{4.2}$$

where N_i is a set of nodes neighboring the *i*-th node, *dist*(*a*, *b*) denotes the Euclidean distance between *a* and *b*; and *d* is an integer value.

In (4.2), a parameter *d* corresponds to the order of the NS, e.g. *d* = 1 defines the NS of order 1 described in (4.5) and *d* = 10 defines the NS of order 7 shown in Figures 4.1 and 4.2. Several values of the parameter *d* with the corresponding orders of NS are given in Table 4.1, which has been computed empirically.

Table 4.1 Correspondence of the NS order to the parameter *d*

NS order	Parameter *d*
1	1
2	2
3	4
4	5
5	8
6	9
7	10

The Euclidean distance in (4.2) is calculated using the following formula for the 2D space between two points P (x_1, y_1) and Q (x_2, y_2):

$$dist(P,Q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{4.3}$$

where x and y are the horizontal and vertical coordinates of a point.

The neighborhood relationship has the following properties:

- A site cannot be a neighbor to itself;
- The neighboring relationship is mutual.

In other words, the neighborhood relationships are symmetric but non-reflexive. The 2D alignment of two strings is viewed as the rectangular area of nodes of character pairs (s_i, t_j) constituting the set of nodes S:

$$S = \{(i, j) \mid 1 \le i \le n, \quad 1 \le j \le m\} \tag{4.4}$$

and a node has up to four neighbors in the NS of the first order:

$$N_{i,j} = \{(i-1, j), \quad (i+1, j), \quad (i, j-1), \quad (i, j+1)\} \tag{4.5}$$

The order of the NS is the measure of the number of nodes. The order of the NS can also be defined as the measure of complexity of the NS. A node corresponds to one square in the 2D lattice. A node has fewer neighbors when it is located in the corner or on the border of the 2D lattice [52].

	-4	-3	-2	-1	0	1	2	3	4
-4	14	13	12	10	9	10	12	13	14
-3	13	11	8	7	6	7	8	11	13
-2	12	8	5	4	3	4	5	8	20
-1	10	7	4	2	1	2	4	7	10
0	9	6	3	1	0	1	3	6	9
1	10	7	4	2	1	2	4	7	10
2	12	8	5	4	3	4	5	8	12
3	13	11	8	7	6	7	8	11	13
4	14	13	12	10	9	10	12	13	14

Figure 4.1 Node rankings for the seventh order neighborhood system of the node (0,0) in the 2D lattice of nodes.

Figure 4.1 shows the neighborhood system of the seventh order for the node (0,0). It contains thirty-four highlighted nodes with values less than or equal to 7. The nodes contain numbers from 1 to 14 corresponding to the outermost neighboring sites in the neighborhood system of the *n*-th order, e.g. the nodes with values less than or equal to 1 belong to the NS of the first order, the nodes with values less than or equal to 2 constitute the NS of the second order and so on.

Consider the node (2, 1) in Figure 4.1. According to formula (4.3), the value of the squared Euclidean distance between this node and the center node (0, 0) is $(2–0)^2 + (1–0)^2 = 5$. This value corresponds to the parameter $d = 5$ following formula (4.2). According to Table (4.1), $d = 5$ means that the node (2, 1) belongs to the neighborhood distance of 4^{th} order. Thus, this explains the number 4 in the cell (2, 1).

The pair $G=(S, N)$ constitutes the graph G where S is a set of nodes and N is a set of edges. A clique c in G is defined as the subset of the neighboring nodes of S such that any two nodes of c are neighbors of each other. The potentials V are assigned to the cliques in order to distinguish different local interactions [51]. The above-mentioned notions are used in the Maximum a Posteriori (MAP) – MRF approach advocated by Geman et al. [53] and others [54, 55]. The MAP-MRF

framework makes it possible to create agile algorithms for different kinds of problems in image recognition, computer vision, and string matching [52].

	-4	-3	-2	-1	0	1	2	3	4
-4	32	25	20	17	16	17	20	25	32
-3	25	18	13	10	9	10	13	18	25
-2	20	13	8	5	4	5	8	13	20
-1	17	10	5	2	1	2	5	10	17
0	16	9	4	1	0	1	4	9	16
1	17	10	5	2	1	2	5	10	17
2	20	13	8	5	4	5	8	13	20
3	25	18	13	10	9	10	13	18	25
4	32	25	20	17	16	17	20	25	32

Figure 4.2 Squared Euclidean distance values for the seventh order neighborhood system of the node (0, 0) on the 2D lattice of nodes.

Figure 4.2 shows the squares of distance values assigned to the nodes used in (4.2) to determine nodes constituting the neighborhood system of the given order. The highlighted cells constitute the NS of the seventh order.

The Euclidean distance formula is applied to the lattice of nodes. The first point P is the point for which the NS is calculated. The point P is treated as the origin of the Euclidean space. The axes are going through the origin: y-axis top-to-bottom and x-axis left-to-right as shown in Figure 4.3. The unit of length is one node. So the node located at the right side from the origin has coordinates (1, 0), the one at the left side (−1, 0), the one above (0, −1), the one below (0,1) and so on (see Figure 4.3). The coordinates of the other point Q are substituted into the formula (4.3). When the inequality specified in (4.2) holds, the point Q belongs to the NS of the point P.

Due to the formula (4.2), the NS always has a symmetric form when the NS is not intersected by the border of the lattice. Figures 4.1 and 4.2 display the NS for a random node of the lattice. The node coordinates that constitute the NS for any node are calculated once and then applied to the rest of the lattice in order to get the NS for

the other nodes. A node, for which an NS is built, is always located at the point with the coordinates (0, 0).

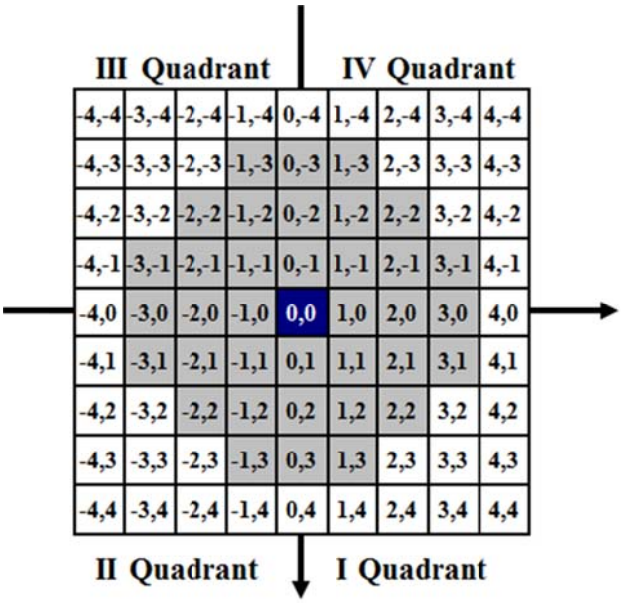


Figure 4.3 The assignment of Euclidean coordinates to the lattice of nodes.

The edit path optimality between two strings $S_{1..n}$ and $T_{1..m}$ depends on the sub-problem optimality of finding the edit paths between the substrings $S_{1..i}$ and $T_{1..j}$ where $i \le n$ and $j \le m$. This is the causal relationship of the edit distance calculation problem for two strings. The causal relationship projected onto the lattice of nodes produces the following assertions: the cause consists of the nodes located in the quadrant II of the ordinary neighborhood system for the given node (Figure 4.3), and the given node is the effect.

The author limits the causal nodes to those which belong to the causal neighborhood system of the given node. Then the causal neighborhood system is defined as follows:

$$N_i = \{i' \in S \mid [dist(p_{i'}, p_i)]^2 \le d, i' \le i\} \tag{4.6}$$

The causal NS is a special case of the neighborhood system where the indices of the neighboring nodes are constrained to be less than or equal to the indices of the

center node. Figure 4.4 shows the location of the causal neighborhood system of the seventh order represented with the highlighted cells in the 2D lattice. The indices are shown as bold numbers outside the lattice.

	-4	**-3**	**-2**	**-1**	**0**
-4	14	13	12	10	9
-3	13	11	8	7	6
-2	12	8	5	4	3
-1	10	7	4	2	1
0	9	6	3	1	0

Figure 4.4 The causal neighborhood system of the seventh order for the node (0, 0) in the 2D lattice of nodes. The numbers n = 1...14 express the outermost neighboring sites in the causal neighborhood system of the n-th order.

	-4	**-3**	**-2**	**-1**	**0**
-4	32	25	20	17	16
-3	25	18	13	10	9
-2	20	13	8	5	4
-1	17	10	5	2	1
0	16	9	4	1	0

Figure 4.5 The causal neighborhood system of the seventh order for the node (0,0) in the 2D lattice of nodes. The numbers indicate the squares of the Euclidean distance $[dist(p_{i'}, p_i)]^2$ from the node (0, 0) to the other nodes.

The pair wise comparison of Figures 4.1 to 4.4 and Figures 4.2 to 4.5 shows that the use of the causal neighborhood system significantly cuts the number of nodes compared with the regular neighborhood system of the corresponding order, thus reducing the problem size and improving the computational time.

Utilizing the above concepts and the MAP-MRF framework, the MRFED is introduced as the measure of similarity between two strings. When the first order causal neighborhood N is adopted, then the neighbors of the (i, j) node are the nodes $(i-1, j)$ and $(i, j-1)$ and the collection of cliques is $C=\{c_1, c_2\}$ where $c_1=\{(i, j), (i-1, j)\}$ with the clique potential V_{c1} and $c_2=\{(i, j), (i, j-1)\}$ and with the clique potential V_{c2}.

The clique potential V_c is the function on S with the property that V_c depends only on those coordinates for which $s \in C$. The resultant objective energy function for the node (i, j) is formulated as

$$MRFED'(i,j) = \min\{MRFED'(i-2,j) + V_{c1}, MRFED'(i,j-2) + V_{c2}\} \quad (4.7)$$

The Needleman–Wunsch [22] edit distance described in the Section 2.4.13 of this work can be considered as a particular case of the MRFED with a second order causal neighborhood system. The clique potentials in the MRFED may be considered as the insertion, deletion and substitution costs of the Needleman-Wunsch algorithm.

The number of cliques increases rapidly with the growth of the neighborhood system order leading to an increase in the computational time. To reduce the problem size, the clique potential values are defined as follows:

$$V_c = \begin{cases} \alpha * (k-1) \text{ if } h(S[i-k+1,i]) = h(T[j-k+1,j]) \\ \infty \text{ otherwise} \end{cases} \quad (4.8)$$

In (4.8), k is the number of nodes in the clique and h is a histogram, which is defined as an associative array counting the number of occurrences of each distinct symbol in a string.

The parameter $\alpha \geq 0$ assigns a weight to the clique potential V_c. When $\alpha = 0$, the MRFED degrades to the Needleman-Wunsch edit distance. The clique potential gets the minimal weight when $\alpha \to 0$. During the experimental phase, the author set α= 0.5 as suggested in Wei's work [51].

Putting together (4.7), (4.8), and (2.16), the final expression for the MRFED takes the form:

$$MRFED_{ij} = \begin{cases} NW_{ij}, \text{ if } h(S[i-k+1,i]) \neq h(T[j-k+1,j]) \\ \min(NW_{ij}, MRFED_{i-k,j-k} + \alpha * (k-1)) \text{ otherwise} \end{cases}, \; k = 2..K \quad (4.9)$$

where k is the number of nodes in the clique, K is the number of diagonal nodes in the 2D lattice representing the causal neighborhood system N, and NW_{ij} is

the Needleman-Wunsch edit distance for the substrings $S_{1..i}$ and $T_{1..j}$. For example, k = 2 for a causal NS of the seventh order as depicted in Figure 4.5. There you can see two grey squares on the main diagonal leading to the node (0, 0).

4.1.2 MRFED Algorithm

Inspired by the Reshuffling Markov Edit Distance [51], the author formulates the Markov Random Field-based Edit Distance and proposes the flexible algorithm for computing the MRFED depicted in Figure 4.6. As proposed in [51], the MRFED uses the notions of neighborhood system, cliques and clique potentials, but the author interprets these notions in a different way, such that the edit distance is minimized at each iteration and then normalized by the edit path length.

Post normalization was implemented in order to improve the results of the MRFED. It is known that post normalization gives worse results compared to normalization in the ordinary sense, but it takes less computational time [25]. The post-normalization was done by storing the edit path and then dividing the MRFED value by the length of the edit path in the last iteration, as seen in formula (4.10). Below, n and m are the lengths of the strings S and T, and $L(P)$ is the length of the edit path from S to T.

$$MRFED_{n,m} = \left\{ \frac{unnormalized\ MRFED_{n,m}}{L(P_{n,m})} \right. \tag{4.10}$$

The MRFED was implemented in Java by extending the open-source Second String project [23]. The MRFED algorithm for calculating the Markov Random Field-based distance for two strings $S_{1..n}$ and $T_{1..m}$ is given below in Figure 4.6.

The algorithm starts with the creation of a placeholder for the MRFED values corresponding to $S_{1..i}$ and $T_{1..j}$ matching substrings. All the intermediate values should

be stored in order to calculate the final MRFED, which corresponds to the whole lengths of strings $S_{1..n}$ and $T_{1..m}$.

In Step 2, the initial values are set. They correspond to the transformation of the empty string λ [25] to a substring and vice versa. The two embedded loops shown in Steps 3 and 4 are required to calculate the Needleman-Wunsch edit distance and clique potentials for the substrings being matched at the current iteration.

0. Initialize k from a user input, set $\alpha = 0.5$.
1. Create the 2D array MRFED[1..n][1..m]
2. Set *MRFED* starting values j**gapCost* for the first row and i**gapCost* for the first column
3. For $i = 2$ to n do
    4. For $j = 2$ to m do
        5. Calculate $NW[i, j]$
        6. Search for the substrings with the equal histograms, setting the length of the compared substrings as $r = min(k, i, j)$. Within the loop, go up to r characters back from the current position.
        For $p = 0$ to r do
            7. If the histograms of the substrings $S_{i-p..i}$ $T_{j-p..j}$ match
            8. Then calculate the clique potential
                $V_{ij}[r\text{-}p] = MRFED[i\text{-}(r\text{-}p), j\text{-}(r\text{-}p)] + \alpha\text{*}(r - p - 1)$
9. Set $MRFED[i, j]$ to min(V[1..k]) and add to the edit path the new traces
10. Set $MRFED[i, j]$ to min($MRFED[i, j]$, $NW[i, j]$)
11. Return $MRFED[n, m]$ divided by the length of the edit path.

Figure 4.6 Description of the MRFED algorithm.

The loop at Step 6 serves to find the equal histograms of the substrings, which stretch from the current position i for r characters back; r is set to min(k, i, j) to prevent exceeding the string boundaries. The parameter k is the number of nodes in the corresponding clique (see formulas (4.8) and (4.9)). The clique potentials V_{ij} are calculated whenever equal histograms are found and *MRFED* is set to the maximum value of V_{ij}. At Step 10, the smaller of the values $MRFED_{ij}$ and NW_{ij} is picked. When the two substrings reach the lengths of the full strings, post normalization is applied and the final MRFED value is returned. The suggested choice for k, used at Step 6, is

in the range 3..6, which provides good practical results and reduces the computational time of the algorithm to $O(nm)$.

Figure 4.7 depicts the MRFED distance matrix computed for the string S "markovdelfie" of length 12 and the string T "markovfield" of length 11. The optimal edit path is depicted with black cells. The blue cells show histogram matching, which lies on the edit path and therefore contributes to the final value. In the edit path, a horizontal move denotes a deletion, a diagonal move is a match or a substitution, and a move down is an insertion.

	λ	m	a	r	v	k	o	d	e	l	f	i	e
λ	0	1	2	3	4	5	6	7	8	9	10	11	12
m	1	0	1	2	3	4	5	6	7	8	9	10	11
a	2	1	0	1	2	3	4	5	6	7	8	9	10
r	3	2	1	0	1	2	3	4	5	6	7	8	9
k	4	3	2	1	1	1	2	3	4	5	6	7	8
o	5	4	3	2	2	2	1	2	3	4	5	6	7
v	6	5	4	3	2	3	1	2	3	4	5	6	7
f	7	6	5	4	3	3	2	2	3	4	4	5	6
i	8	7	6	5	4	4	3	3	3	4	5	4	5
e	9	8	7	6	5	5	4	4	3	4	5	5	4
l	10	9	8	7	6	6	5	5	4	3	4	4	5
d	11	10	9	8	7	7	6	5	5	4	4	5	5

Figure 4.7 The MRFED distance matrix and the 2D word alignment.

4.2 Shortest Path Edit Distance

4.2.1 Motivation

As shown in Section 4.1, the main problem of the MRFED is its low speed, which prevents its use for computational tasks, where datasets of tens of thousands of records are involved. Also, the performance of the MRFED in terms of average precision, maximum F_1, and precision-recall curves is not ideal compared to other existing methods. The present research has been conducted to develop a new method

that preserves the best features of the previous work on MRFED, while contributing several innovations and improvements to it.

4.2.2 Lattice of String Neighborhoods

In the first step of the modeling process, a lattice of nodes representing substring interactions is constructed. The substrings are denoted as $S_{i\text{-}k\ldots i}$ and $T_{j\text{-}k\ldots j}$ where $k = 0\ldots\min(n,m)$ and n and m denote the lengths of the strings S and T. The set of string neighborhoods will be defined as a set C of the consequent substrings of the length k, $k = 1\ldots n$ for ($\lfloor n/k \rfloor - 1$) elements and the $\lfloor n/k \rfloor$-th element is of length $n-$ ($\lfloor n/k \rfloor - 1$) $\cdot$ k. Each neighborhood will be given a number in the range $1\ldots\lfloor n/k \rfloor$. In Figure 4.8 (a) for string S = "Albert Einstein", n = 15, k = 2, therefore $\lfloor n/k \rfloor = 7$.

(a)

	A	l	b	e	r	t		E	i	n	s	t	e	i	n
A	1.5		2		2		2		2		2		3		
r	2		2		2		2		2		2		3		
c	2		2		2		2		2		2		3		
h	2		2		2		2		1.5		2		3		
i	2		2		2		2		2		2		2.5		
m	2		2		2		2		2		2		2.5		
e	2		2		2		2		2		2		2.5		
d	2		2		2		2		2		2		2.5		
e	2		2		2		2		2		2		2.5		
s	2		2		2		2		2		2		2.5		

(b)

	A	l	b	e	r	t		E	i	n	s	t	e	i	n
A	2.67			3			3			3			3		
r															
c															
h	3			3			3			3			3		
i															
m															
e	3			2.67			3			3			2.67		
d															
e															
s															

Figure 4.8 Lattices computed for the strings "Albert Einstein" and "Archimedes" using the string neighborhoods of lengths two (a) and three (b).

A lattice element is defined as an interaction of two string neighborhoods, where one string neighborhood belongs to string S and another to T. Two examples of typical lattices are given in Figure 4.8. Consider two border conditions: k = 1 and k =

min(n, m). In the simplest case, k = 1, thus excluding the substring matching operation from consideration and degrading the model to the classic case of character-to-character operations. In that case, a lattice node represents the interaction between the pair of characters S_i and T_j.

When k = min(n, m), three sub-cases are possible depending on the lengths of the strings S and T:

- $n = m$ leads to a lattice consisting of the single node, which includes a comparison operation of two whole strings.
- $n > m$ gives a two node lattice where the first node matches substrings $S_{1..m}$ with $T_{1..m}$ and the second node $s[(m\text{-}n)..n]$ is matched with an empty string.
- $m > n$ will give a two node lattice where the first node is $S_{1..n}$ matching with $T_{1..n}$ and the second node is the empty string matching with $T_{(n\text{-}m)..m}$.

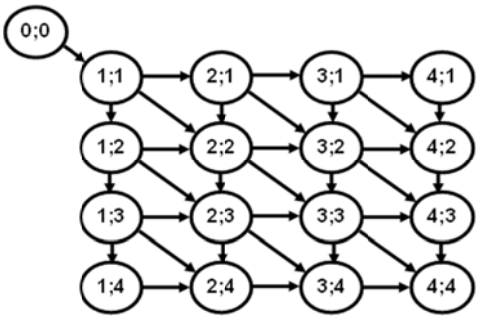
The lattices calculated for the strings "Albert Einstein" and "Archimedes" are shown in Figure 4.8. Figure 4.8 (a) depicts the node sizes and values calculated for the string neighborhood length k = 2. The partition that results, when the length k = 3, is shown in Figure 4.8 (b). The values of the nodes are calculated using the arithmetic mean method, defined below in this chapter.

4.2.3 Lattice-based Graph Composition

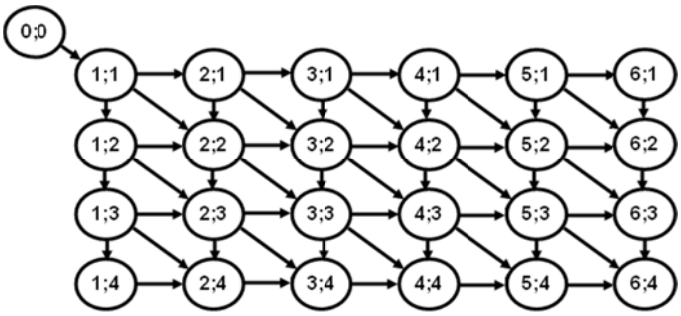
The second step of the proposed Shortest Path Edit Distance (SPED) algorithm is the transformation of the lattice into a directed, weighted acyclic graph. In the classic case, the two strings being matched are put onto the vertical and horizontal sides of the matrix, which is filled by the values obtained at every iteration of the string distance calculation. Then the edit path can be shown as a sequence of cells, starting at the cell corresponding to the first characters of each string to the cell located at the intersection of the last characters. The last cell contains the value of the edit distance between the two strings. Any path from the first to the last cell can have horizontal,

vertical, and/or diagonal movements. Usually, the horizontal direction is interpreted as the deletion, vertical as the insertion and diagonal as the substitution or match of a pair of characters.

(a)



(b)



(c)

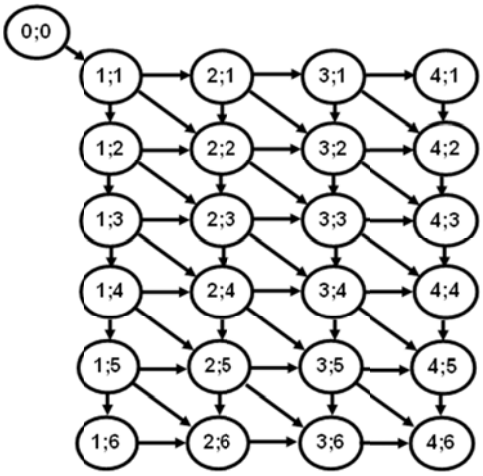


Figure 4.9 Graphs constructed from the lattices of (a) 4x4, (b) 6x4, (c) 4x6 dimensions.

Each lattice cell is converted into a graph vertex. As shown in Figure 4.9, each vertex with the coordinates in the range [2..*n,* 2..*m*] has three incoming edges: horizontal, vertical, and diagonal. All the vertices from the top row down except (1, 1) have horizontal incoming edges. The vertices from the left column only have vertical

incoming edges. The source node is added at the left top corner of the graph. It does not have any incoming edges and is connected to the vertex (1, 1) of the graph by a diagonal edge, which is the single incoming edge of the vertex (1, 1). The horizontal and vertical edges are assigned a gap cost. During the experimental stage, the author used the gap cost 1 following a common approach [28]. The diagonal edge is assigned a value stored in a lattice cell. The process of weight assignment is described in Sections 4.2.2 and 4.2.9. As described above, this value is the weight of the string neighborhood edit operation. The source vertex will be used as a placeholder for the starting point of the algorithm.

Figure 4.9 depicts three possible cases of the graph shape: case (a) when strings S and T are of the same length, $n = m$; (b) when string S is longer than T, $n>m$; and (c) when string S is shorter than T, $n<m$. These three cases are important because they influence the graph traversal pattern as described below in Section 4.2.4.

4.2.4 Analysis of Shortest Path Graph Algorithms

In the SPED algorithm, the task of calculating a string distance value between two strings becomes a task of calculating the shortest path from the source vertex to the destination vertex. The destination vertex corresponds to the pair of last string neighborhoods of strings S and T. By design, the graph is a directed, weighted, acyclic graph. The most efficient algorithm to solve the shortest path problem should be chosen to find an optimal solution. The classical algorithms are well known:

- Dijkstra's algorithm—the single-source shortest path for graphs with non-negative edges, $O(|V|^2)$ [60];
- Floyd's algorithm—the all-pairs shortest path for the weighted directed graph, $O(|V|^3)$ [61];
- Bellman-Ford's algorithm—the single-source shortest path that can be used for a weighted graph allowing the edge weights to be negative, $O(|V|\cdot|E|)$ [62, 63].

The two most efficient algorithms that work for directed acyclic graphs and allow negative edge weights are the Reaching algorithm [64] and the Pulling algorithm [59], both working in $O(n)$ time.

4.2.5 Reaching and Pulling Algorithms

The Reaching and Pulling techniques shown in Table 4.2 are similar to other dynamic programming algorithms. These methods assign labels to graph vertices first and then traverse a graph from the smallest to the largest label, computing the shortest path from node 1 to node k at the k-th iteration. The only difference is that the Pulling algorithm examines incoming edges, while the Reaching algorithm operates with emanating edges.

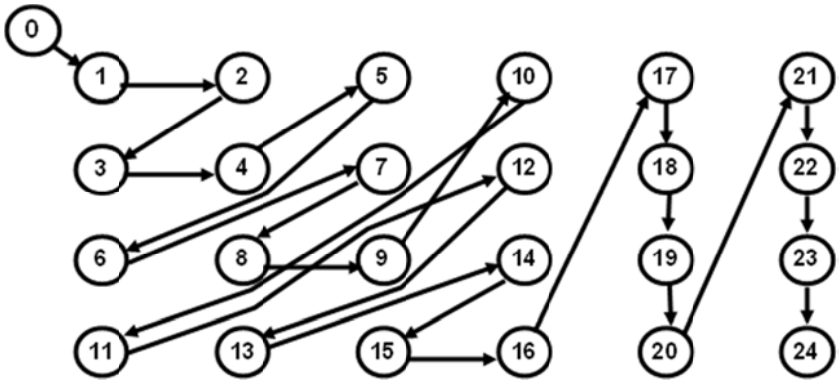
Table 4.2 The Reaching and Pulling algorithms

The Pulling Algorithm	**The Reaching Algorithm**
1. Assign labels to graph nodes.	
2. Process the vertices from the lowest to the highest label.	
3. At the k-th iteration, find $\min(d(i) + c_{ik})$ for all incoming edges (i, k)	3. At the k-th iteration, find $\min(d(k) + c_{kj})$ for all emanating edges (k, j)

By design, each lattice node is transformed into a graph vertex. The weights of incoming diagonal edges are set to the lattice values of the corresponding vertices. Since the SPED algorithm uses the incoming edges, the choice of the Pulling algorithm is clearly better because it operates with the costs of incoming edges.

The first step of the Pulling algorithm requires a preprocessing phase. Vertices of the graph must be marked with labels from 1 to $|V|$ with respect to the subsequent calculations. The direction of the emanating edges should be taken into consideration when assigning a label. The labels are set in a zigzag pattern for every vertex of the graph in a way that does not leave behind any unlabeled vertex.

(a)



(b)

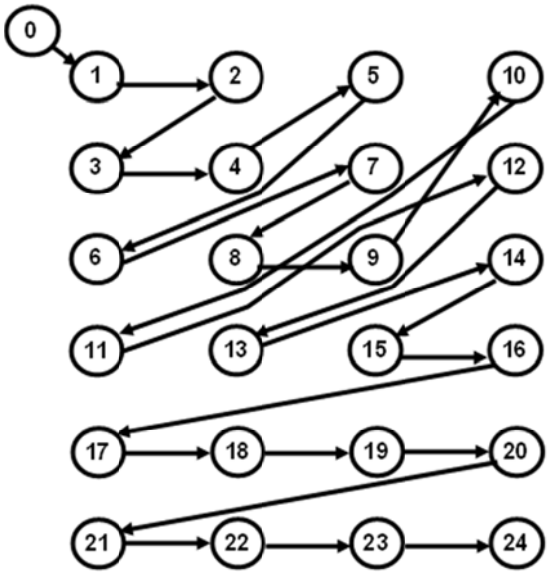


Figure 4.10 Label assignment to the graph nodes.

For the 6x4 lattice depicted in Figure 4.9 (b), the labels are assigned to the nodes in the following order: (0, 0) → (1, 1) → (2, 1) → (1, 2) → (2, 2) → (3, 1) → (1, 3) → (3, 2) → (2, 3) → (3, 3) → (4, 1) → (1, 4) → (4, 2) → (2, 4) → (4, 3) → (3, 4) → (4, 4) → (5, 1) → (5, 2) → (5, 3) → (5, 4) → (6, 1) → (6, 2) → (6, 3) → (6, 4).

Depending on the lattice shape, which is defined by the lengths n and m of the strings S and T, the three patterns of the traversal are possible:

1. The trivial pattern when $n = m$;
2. The Right Side First pattern when $n > m$;
3. The Bottom Side First pattern when $m > n$.

The pattern (2) is shown in Figure 4.10 (a). Here, the right side moves are from node 16 to node 17 and from node 20 to node 21. The pattern (3) is shown in Figure 4.10 (b). Here, the down side moves are from 16 to 17 and from 20 to 21. The trivial pattern (1) is a subset of (2) and (3). For each lattice 6x4 (1) and 4x6 (2) shown in the figures, the trivial pattern is the sub-section of the path that starts at vertex 0 and stops at vertex 16.

4.2.6 Winkler-like Re-scorer

In the last step of the SPED algorithm, its value is adjusted by applying the Winkler-like re-scorer as follows. The two strings are checked for the presence of a common prefix. When several successive initial characters of both strings match, the SPED value for these strings is computed as shown in formula (4.11):

$$SPED = SPED' - prefLength \cdot 0.1 \cdot (1 - SPED') \qquad (4.11)$$

where *SPED*' is the value of the SPED algorithm before the application of the Winkler-like re-scorer, *prefLength* is the length of the common prefix, and *SPED* denotes the final score for the two strings.

The original re-scorer by Winkler examined match or non-match of the four initial characters of both strings. In the case of SPED, the re-scorer does not stop at the fourth character. The Winkler-like re-scorer proceeds up to the 100^{th} character, unless there is a mismatch or the end of one of the strings is encountered.

4.2.7 SPED Algorithm Complexity

The complexity of the new algorithm is estimated as $O(n^2)$. Its main steps are shown in Table 4.3. Even though it has the same computational time as the MRFED algorithm, the SPED algorithm shows a faster performance in practice. It is possible

that two algorithms with the same worst-case time complexity have actual run times differing by a constant.

Steps 3 and 4, shown in Table 4.3 can be viewed as the classic case of the edit distance calculation:

- The shortest path from the source to the destination can be considered as the edit path between two strings;
- The cost of the shortest path is the value of the distance function.

Table 4.3 The SPED Algorithm complexity

Step	Description	Complexity
1	Calculation of the lattice nodes.	$O(n^2)$
2	Transformation of the lattice into the graph and calculation of the edge costs.	$O(n^2)$
3	Finding the shortest path from the source vertex to the destination vertex.	$O(n)$
4	Application of the Winkler-like re-scorer.	$O(n)$
5	Returning the shortest path cost, which is the value of the edit distance.	$O(1)$
Total		$O(n^2)$

The usual way to depict the intermediate edit distance values and the trace back, showing the path of the edit distance calculation, is shown in Figure 4.11. Each cell shows the intermediate value of the SPED, computed for the corresponding substrings of the two strings "Albert Einstein" and "Archimedes." The arrows portray the trace back, which is the reversed edit path for the two strings.

	Al	b e	r t	E	i n	s t	e i n
A r	1.5	3.5	5.5	7.5	9.5	11.5	14.5
c h	3.5	3.5	5.5	7.5	9.5	11.5	14.5
i m	5.5	5.5	5.5	7.5	9.0	11.0	14.0
e d	7.5	7.5	7.5	7.5	9.5	11.0	13.5
e s	9.5	9.5	9.5	9.5	9.5	11.5	13.5

Figure 4.11 The interim SPED values and the trace back for the algorithm with the string neighborhood of length 2.

4.2.8 Parameters Adjusting Performance

It is possible to fine-tune the SPED by changing its internal parameters of the two following types:

- The internal string distance which assigns weights to the string neighborhood edit operations. The internal metric can be selected as an existing string edit distance or as a new operation assigning values as the result of a similarity estimation of its arguments.
- The length of the string neighborhood. This parameter allows adapting to the existing local dependencies in the strings, which may be unique to a given dataset.

4.2.9 String Neighborhood Edit Operations Assignment

The string neighborhood edit operations assign a weight to a pair of string neighborhoods. It is possible to use an existing string distance metric for weight assignment. In this work, the author uses the Arithmetic Mean method, described as follows. Within two corresponding string neighborhoods, each pair of corresponding characters is assigned the value 0 for a match and 1 for a mismatch. Then these values are added and the sum is divided by the number of pairs. Consider the example of the weight calculation for the two string neighborhoods "si" and "ki":

- The pair ("s", "k") is assigned the weight of 1;
- The pair ("s", "i") is assigned the weight of 1;
- The pair ("s"; "k") is assigned the weight of 1;
- The pair ("i", "i") is assigned the weight of 0;
- Adding these four weights together, the sum of 3 is obtained;
- The sum of 3 is divided by the number of pairs, which is 4;
- Thus the weight for the string neighborhoods "si" and "ki" is 0.75.

4.3 Histogram Difference Method

The Histogram Difference (HD) method was inspired by Wei's work [51] where a histogram difference was involved at one of the steps in the Coherence Markov Edit Distance algorithm. The histogram is defined as an associative array, counting the number of occurrences of each character in a string. The implementation of the histogram in the Java programming language is done utilizing the hash table data structure. Wei used formula (4.12) to compute a HD between two strings *S* and *T*. In (4.12), *hist* is the histogram function [51], i.e. the value of the hash table for this argument:

$$hist(s) - hist(t) = \sum (0.5 \cdot (hist(S) \cup hist(T)) - hist(S) \cap hist(T)) \qquad (4.12)$$

In this work, a new definition of the HD is introduced in formula (4.13) below:

$$histDiff(S,T) = \sum (hist(S) \cup hist(T) - 2(hist(S) \cap hist(T))) \qquad (4.13)$$

The example of the application of formulas (4.12) and (4.13) is shown in Figure 4.12.

$h_S = \begin{bmatrix} a & 2 \\ b & 7 \\ c & 5 \end{bmatrix}$, $h_T = \begin{bmatrix} b & 3 \\ c & 4 \\ d & 8 \end{bmatrix}$.

Following (4.12):

$$\Delta h_1 = \sum (\frac{1}{2} \begin{bmatrix} a & 2 \\ b & 10 \\ c & 9 \\ d & 8 \end{bmatrix} - \begin{bmatrix} b & 3 \\ c & 4 \end{bmatrix}) = \sum (\begin{bmatrix} a & 1 \\ b & 5 \\ c & 4.5 \\ d & 4 \end{bmatrix} - \begin{bmatrix} b & 3 \\ c & 4 \end{bmatrix}) = \sum \begin{bmatrix} a & 1 \\ b & 2 \\ c & 0.5 \\ d & 4 \end{bmatrix} = 7.5$$

Following (4.13):

$$\Delta h_2 = \sum (\begin{bmatrix} a & 2 \\ b & 10 \\ c & 9 \\ d & 8 \end{bmatrix} - 2\begin{bmatrix} b & 3 \\ c & 4 \end{bmatrix}) = \sum (\begin{bmatrix} a & 2 \\ b & 10 \\ c & 9 \\ d & 8 \end{bmatrix} - \begin{bmatrix} b & 6 \\ c & 8 \end{bmatrix}) = \sum \begin{bmatrix} a & 2 \\ b & 4 \\ c & 1 \\ d & 8 \end{bmatrix} = 15$$

Figure 4.12 Two approaches to HD calculation.

Note that $\Delta h_2 = 2 \cdot \Delta h_1$. It may seem that string distance values based on (4.12) and (4.13) only differ by a factor of 2. Nevertheless, when Δh_1, Δh_2, $|S|$, and $|T|$ from the example above are substituted in (4.14) below, $HD_1 = 0.74$ and $HD_2 = 0.48$ are derived for cases (4.12) and (4.13), i.e., different final results are obtained, with (4.13) giving the better one.

Figure 4.12 showed the values produced by formulas (4.12) and (4.13). The formula (4.13), introduced in this thesis, is more suitable for the HD function. It is advocated below utilizing a Venn diagram. Consider histograms h_S and h_T as two sets A and B having certain elements in common, as depicted in Figures 4.13 and 4.14. For the purpose of finding a string dissimilarity value, the application of the HD function to the histograms A and B should return a subset equal to $A' \cup B'$ or, in words, the symmetric difference of the sets A and B. Figure 4.6 depicts it as four separated subsets. Now it is easy to see that in order to obtain $A' \cup B'$, formula (4.13) should be applied, whereas formula (4.12) does not produce the required value.

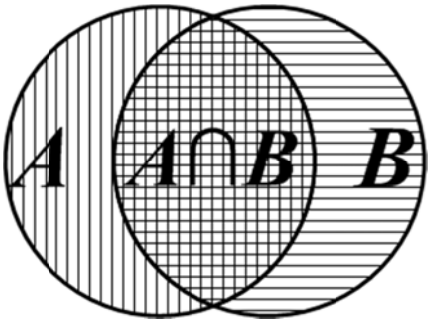


Figure 4.13 Venn diagram view of the HD.

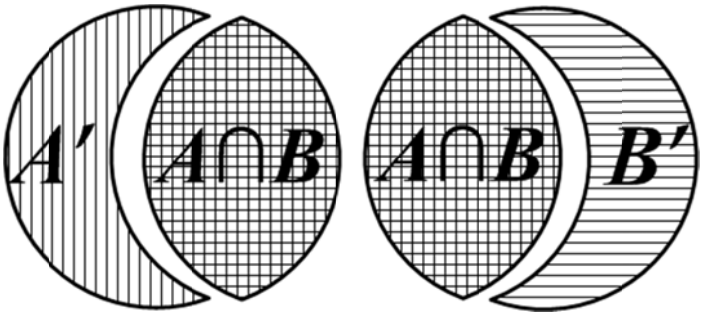


Figure 4.14 Separated subsets view of the HD.

The formula of the HD method takes the form:

$$HD = \frac{histDiff(S,T)}{\sum(hist(S)+hist(T))} \tag{4.14}$$

The *histDiff* (*S*,*T*) function is normalized by the sum of the histograms of the strings *S* and *T*. This is done in order to take into account the lengths of both strings. The ratio is subtracted from 1 in order to comply with the specifications of the Second String project design. Now, when *S* equals *T*, the *histDiff* (*S*,*T*) becomes 0 and *HD* becomes 1. When *S* and *T* don't have any characters in common, *histDiff* (*S*,*T*) becomes equal to $\sum(hist(S)+hist(T))$ thus the numerator is equal to the denominator and the ratio becomes equal to 1, making *HD* equal to 0.

The computational complexity of the HD method is $O(n)$ as shown in the table below.

Table 4.4 The HD **A**lgorithm complexity

Step	**Description**	**Complexity**
1	Calculation of *hist*(*S*)	$O(n)$
2	Calculation of *hist*(*T*)	$O(n)$
3	Calculation of $hist(S) \cap hist(T)$	$O(1)$
4	Summation operations	$O(1)$
5	Subtraction operations	$O(1)$
6	Deletion operations	$O(1)$
Total		$O(n)$

Several modifications of the HD method are introduced in this work. All the modifications use HD as a core method but apply difference re-scorers. In previous work, the SPED method used the Winkler-like re-scorer. The idea of adjusting the final string distance value gets broader implementation in the family of HD methods discussed below.

4.3.1 HD with Normalized Smith-Waterman Re-scorer

This variant of the HD method uses the Smith-Waterman re-scorer normalized by the sum of the lengths of strings *S* and *T*. The initial HD method value is calculated using the formula (4.14). In the next step, the Smith-Waterman function is applied to the strings *S* and *T* and the obtained value is divided by the sum of the string lengths, $|S|+|T|$. The normalized Smith-Waterman value is weighted and added to the HD value. These steps are depicted by the formula (4.15) below:

$$HDSW = HD(S,T) + (1 - HD(S,T))\frac{SW(S,T)}{|S| + |T|} \tag{4.15}$$

where *HDSW* is the HD method with the normalized Smith-Waterman Re-Scorer, *HD* is the Histogram Difference function, and *SW* is the Smith-Waterman function. The implementation of the re-scorer shown in (4.15) ensures that the HDSW value lies within the [0, 1] interval.

The computational complexity of the HDSW method is $O(n^2)$. The analysis behind it is as follows: The HD complexity is $O(n)$; the complexity of the summation, subtraction, and deletion operations are $O(1)$ each; the complexity of the Smith-Waterman method is $O(n^2)$.

4.3.2 HD with TFIDF and Jaccard Re-scorers

This modification of the HD method utilizes two re-scorers: TFIDF and Jaccard. As in Section 4.3.1, the HD method is computed for the strings *S* and *T*. Then TFIDF and Jaccard re-scorers are applied sequentially, as show in the formulae (4.16) and (4.17) below.

$$HDTF = HD(S,T) + (1 - HD(S,T)) \cdot TFIDF(S,T) \tag{4.16}$$

$$HDTFJ = HDTF(S,T) + (1 - HDTF(S,T)) \cdot Jaccard(S,T) \tag{4.17}$$

where *HDTF* is the HD method after the application of the TFIDF re-scorer, *HD* is the Histogram Difference function, *HDTFJ* is the HD method with both the TFIDF and the Jaccard re-scorers applied, and *Jaccard* is the Jaccard function. The weighted application of the re-scorers guarantees that the *HDTFJ* value falls within the interval [0, 1].

The computational complexity of the HDTFJ method consists of the computational complexities of the HD method and the re-scorers contributing to the final value. The HD complexity is $O(n)$, the complexity of the summation and subtraction operations is $O(1)$ each. The Jaccard method complexity is $O(D_{S,T})$, where $D_{S,T}$ is the number of individual terms in the strings S and T. The TFIDF method complexity is $O(N_R + D_R)$, where N_R is the number of records in a dataset and D_R is the total number of individual words in a dataset. Since $D_R > D_{S,T}$, the total complexity of the HDTFJ method is $O(n + N_R + D_R)$.

4.3.3 HD with the Longest Common Prefix and TFIDF Re-scorers

The Histogram Difference with the Longest Common Prefix Re-scorer (HDLCP) method uses the weighted normalized longest common prefix length to adjust the string similarity value. The author had successfully applied the Winkler-like re-scorer in the past [96]. The HDLCP is the logical continuation of the previous effort, which had only used the first few characters to re-score the final value.

The HDLCP utilizes the length of the longest common prefix, which is normalized by the length of the shortest string in the pair, $\min(|S|,|T|)$. After applying the longest common prefix re-scorer, the TFIDF re-scorer is employed. The formulae for HDLCP are given below:

$$HDLCP' = HD(S,T) + (1 - HD(S,T))\frac{|LCP|}{\min(|S|,|T|)} \tag{4.18}$$

$$HDLCP = HDLCP'(S,T) + (1 - HDLCP'(S,T)) \cdot TFIDF(S,T) \quad (4.19)$$

where *HDLCP'* is the HD method re-scored with the longest common prefix re-scorer, *HD* is the Histogram Difference function, and *LCP* is the longest common prefix. *HDLCP* is the final value of the HDLCP method after applying both re-scorers, and *TFIDF* is the TFIDF function. The formula for *LCP* is as follows:

$$LCP = \begin{cases} i \text{ IF } S_{1..i} = T_{1..i} \text{ AND } S_{i+1} \neq T_{i+1} \text{ AND } i < \min(|S|,|T|) \\ \min(|S|,|T|) \text{ IF } S_{1..i} = T_{1..i} \text{ AND } i = \min(|S|,|T|) \end{cases} \quad (4.20)$$

In (4.18), the length of LCP is weighted and normalized to make sure the value of HDLCP falls in [0, 1]. In (4.16), the *TFIDF* value is weighted in a similar way to (4.19).

The complexity of the HDLCP method is the combined complexities of the LCP and TFIDF algorithms. The LCP complexity is $O(n)$ since in the worst case it stops when the end of the shorter string is reached. The TFIDF method complexity is $O(N_R + D_R)$, where N_R is the number of records in a dataset and D_R is the total number of individual words in a dataset. Thus the total complexity of the HDLCP method is $O(n + N_R + D_R)$.

4.3.4 HD with the Unweighted Longest Common Prefix Re-scorer

The Histogram Difference with the Unweighted Longest Common Prefix Re-scorer (HDULCP) method is similar to the HDLCP method presented in Section 4.3.3. In the HDULCP case, the longest common prefix is weighted differently, namely in such a way that it does not guarantee the final value of the HDULCP to be in the [0, 1] interval. The idea behind the use of the unweighted re-scorer is to give the re-scorer more impact on the final value. The formulae (4.21) and (4.22) define the HDULCP function.

$$HDULCP' = HD(S,T) + HD(S,T)\frac{|LCP|}{\min(|S|,|T|)} \quad (4.21)$$

$$HDULCP = HDULCP'(S,T) + (1 - HDULCP'(S,T)) \cdot TFIDF(S,T) \quad (4.22)$$

The HDULCP computational complexity is calculated in the same way as for the HDLCP method. It is $O(n + N_R + D_R)$.

4.4 Longest Approximately Common Prefix Method

The Longest Approximately Common Prefix (LACP) method uses approximate histogram matches of prefixes to determine the similarity value of a pair of strings. Non-exact matching is performed via the Approximate Histogram Match (AHM) function (formula (4.23)). This function returns "true" when the histogram difference between prefixes of strings *S* and *T*:

- at position (i–1) is less than a threshold parameter α;
- at position i is equal to α or the end of the shorter string is reached, i.e. i=min(|*S*|, |*T*|).

$$AHM(S,T,i) = \begin{cases} true & \text{when} \quad \begin{array}{l} prefHistDiff(S_{1..i-1}, T_{1..i-1}) < \alpha \text{ AND} \\ (prefHistDiff(S_{1..i}, T_{1..i}) = \alpha \text{ OR} \\ i = \min(|S|, |T|)) \end{array} \\ false & \text{otherwise} \end{cases} \quad (4.23)$$

where *prefHistDiff* is a prefix histogram difference function and α is the threshold parameter. At the i-th character position, the histogram difference takes the form:

$$prefHistDiff(S_{1..i}, T_{1..i}) = i - |hist(S_{1..i}) \cap hist(T_{1..i})| \quad (4.24)$$

where *hist* is the histogram function and *i* denotes the prefix length. The number of characters, which are common to the histograms of both prefixes, is subtracted from *i*. Then, the difference is the number of non-common characters.

The AHM function is applied sequentially to a pair of substrings $S_{1..i}$ and $T_{1..i}$, $i \in [1..\min(|S|, |T|)]$. This search stops when the AHM function returns false at some position *i* or the last character of the shorter string is reached. Now, the position *i* is the last character of the longest common prefix. Since the search is started at the first characters of both strings, it follows that the value *i* denotes the length of the longest approximately common prefix. A threshold value α denotes the number of allowed mismatches in the histograms of the prefixes $S_{1..i}$ and $T_{1..i}$.

At the next step, the length of the LACP is normalized by the average length of strings *S* and *T* to ensure that the LACP method value fits in the [0..1] interval. It is also possible to normalize by the length of the shorter string min (|*S*|, |*T*|) to assure a zero-to-one interval of method values.

The formula for the LACP method is as follows:

$$LACP = \frac{|\ prefix\ |}{(|\ S\ | + |\ T\ |) / 2} \tag{4.25}$$

where |*prefix*| is the length of the longest approximately common prefix and *LACP* is the value of the LACP method. The formula for |*LACP*| is given below:

$$|\ prefix\ | = \{i \text{ when } \mathrm{AHM}(S, T, i) = true\} \tag{4.26}$$

where *i* denotes the prefix length.

The expression $hist(S_{1..i}) \cap hist(T_{1..i})$ denotes the intersection of the histograms of two substrings. Let's consider the example given in Figure 4.14.

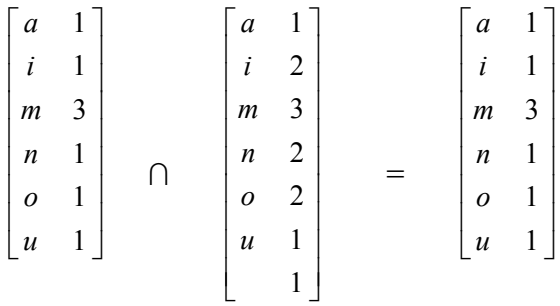


Figure 4.15 Example of the histogram intersection for two UMLS terms: (a) "ammonium", (b) "ammonium ion", (c) the resulting histogram intersection.

In order to get the size of the histogram intersection, the numbers in the resulting matrix are added together. For the case shown in Figure 4.15 (c), the size is (1+1+3+1+1+1)=8.

Table 4.5 Common Prefixes in the UMLS terms

#	String	Length
S_1	Ammonium	8
S_2	Ammonium_ion	12
S_3	AMMONIUM CHLORIDE 1 MG / CYANOCOBALAMIN 5 MCG / FERRIC AMMON IUM CITRATE 40 MG / FOLIC ACID 1 MG / LYSINE HYDROCHLORIDE 100 MG / MAGNESIUM SULFATE 1 MG / MANGANESE SULFATE ANHYDROUS 1 MG / NIACIN 5 MG / PANTHENOL 1 MG / POTASSIUM SULFATE 1 MG / PYRIDOXINE HYDROCHLORIDE 0.5 MG / RIBOFLAVIN 1.2 MG / THIAMINE HYDROCHLORIDE 12 MG / ZINC SULFATE 1 MG ORAL LIQUID [HEMERGON]	370

The normalization by the average is chosen to take into account the lengths of both strings. Consider the example of two pairs of strings sharing the same LACP shown in the Table 4.5. Strings (1) and (2) comprise the first pair, strings (1) and (3)– the second pair. When the prefix length is normalized by the length of the shorter string in a pair, the greater degree of dissimilarity in the pair (1) and (3) is neglected.

Thus with the choice of the normalization by the shorter string length, strings (1) and (2) receive the same dissimilarity value as strings (1) and (3). This is obviously not the desired outcome. However, when normalized by the average string length, the LACP method takes into account the string lengths.

Table 4.5 depicts three UMLS terms. Strings S_1 and S_2 are associated with the same concept with CUI C0002611, the string S_3 belongs to a different concept with CUI C1816069. In the string S_2, the space between the words ammonium and ion was replaced with the underscore character "_" for the sake of presentation. According to formulas (4.23)-(4.25) and the choice of $\alpha = 3$, $LACP(S_1, S_2) = 8 / ((8 + 12)\cdot 0.5) = 0.8$, similarly $LACP(S_1, S_3) = 8 / ((8 + 370)\cdot 0.5) = 0.042$, and $LACP(S_2, S_3) = 10 / ((12 + 370)\cdot 0.5) = 0.052$.

4.4.1 LACP Method Algorithm

The LACP algorithm is shown in Figure 4.16 below. The *prefHistDiff* function was described in detail in the previous section. The LACP algorithm flow is straightforward. It is of linear-time complexity.

```
1     For i=1 to min(|S|, |T|)
2       Begin
3         If prefHistDiff (S1..i, T1..i) = α Then return i
4       End
5     Return min(|S|, |T|)
```

Figure 4.16 Algorithm of the LACP method.

4.4.2 LACP Method Complexity

The worst-case complexity of the LACP algorithm is shown schematically in Table 4.6.

Table 4.6 Complexity of the LACP method

Step	Complexity
Search for the ACP	$O(n)$
Calculation of the *prefHistDiff* function	$O(n)$
Return the final value	$O(1)$
Total complexity	$O(n)$

With the $O(n)$ worst-case complexity, the LACP method performed faster compared to other well-known methods in every experiment conducted. These and other results will be shown below in Chapter 5, Evaluation.

CHAPTER 5

EVALUATION

5.1 Experimental Environment

5.1.1 Benchmark Suite

The open-source Java toolkit Second String [28] was used as the experimental test bed. Second String consists of several Java packages, which implement blockers, string distances, and a routine to perform experiments.

The blockers are needed to reduce the problem size of duplicate detection or clustering tasks. The blockers are methods which remove a large portion of approximately dissimilar record pairs from consideration and leave a minimal subset of approximately similar records in a given dataset [97, 98]. The initial problem size is measured by the total number of record pairs in a given dataset. As an example, the Cartesian product of a dataset with 5,000 records produces $2.5 \cdot 10^7$ pairs of records. A problem size of this kind makes it impossible to perform evaluation in a reasonable amount of time. This is rectified by using blockers.

Second String includes implementations of well-known string similarity metrics such as the distance metrics used in the evaluation described in Section 2.4: Jaccard, Jaro, Jaro-Winkler, Levenshtein, Monge-Elkan, Needleman-Wunsch, Smith-Waterman, Soft TFIDF, and TFIDF. Several datasets are also included in the toolkit. The following datasets were used in the experiments: Animals, Birds, Census, Parks, and Restaurants. The routine to perform experiments implements the benchmarking methodology described in detail below in Section 5.1.2.

5.1.2 Benchmarking Methodology

The Second String toolkit performs an experiment according to a "scenario" specified in the main script file. An example of such a file is shown below in Figure 5.1. The experiment "scenario" consists of several parts:

- One blocker method file name;
- One or more edit distance file names;
- One or more dataset file names;
- Options specifying measures to compute during the experiment.

An experiment starts by reading the instructions in the "scenario" file. The specified blocker is used to build a candidate record set for matching, which is a subset of an original dataset. A candidate set is built for each of the listed datasets in the script.

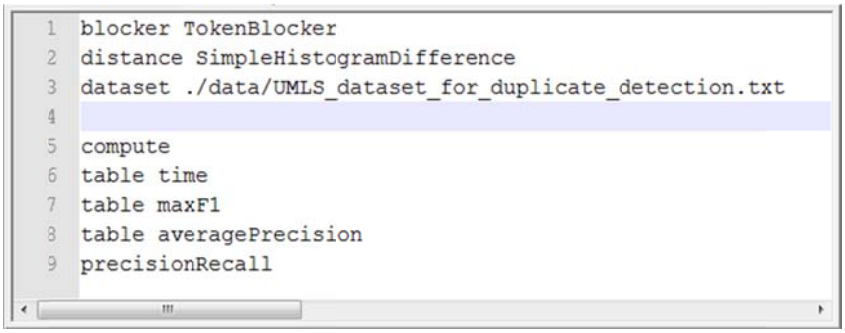


Figure 5.1 Sample experiment "scenario" file.

In the experiment, a blocker provided by Second String was used since the blocking algorithms are beyond the scope of this work. The blocker finds pairs of records containing the same tokens. These pairs are added to the candidate set, until it grows to the upper limit of the number of candidate pairs for a particular token. The upper limit is equal or smaller than the total number of records in the dataset. Each pair of records is marked as correct when both records have the same ID. Otherwise, it is marked as incorrect. A short extract from the candidate set of the Parks dataset is depicted in Table 5.1.

Table 5.1 Record Pairs from the Candidate Set of the Parks Dataset

Record 1	Record 2	Correct Pair Marker
Richmond NB Park	Richmond NBP	True
Sequoia and Kings Canyon NP	Sequoia & Kings Canyon NP	True
Catocin Mtn. Park	Catoctin Mountain Park	True
Jean Lafitte NHP & NPRES	Jean Lafitte NHP & Preserve	True
Colorado NM	Coronado NM	False
Acadia NP	Arches NP	False
Bighorn Canyon NRA	Bryce Canyon NP	False

When the blocking process is completed, a duplicate detection or clustering experiment is performed using each of the selected string similarity functions. The type of the experiment is determined by applying either the clustering or duplicate detection blocker.

Second String assigns a string similarity value to each pair of records in the matching set. When the experiment is completed, the values of the measures selected in the "scenario" file are computed and displayed for analysis.

During the evaluation, four available measures in the Second String toolkit were selected for computation: execution time, average precision, maximum F_1, and precision-recall data used for plotting precision-recall charts.

A stemming algorithm was applied to the dataset records during the experiments. The strings were stripped of the leading and trailing spaces, multiple repeating spaces were replaced with a single space, and the strings were converted to lower case.

5.2 Evaluation of Duplicate Detection in Life and Social Sciences Data Sources

To compare the obtained results, the author built tables aligning average precision, maximum F_1, and execution time for the evaluated metrics. Higher values of average precision and maximum F_1 gained by a string similarity metric indicate better performance. Lower values for execution time also indicate a better result. On the precision-recall curves, better metrics show higher values of precision at more recall points.

In this paper, the author selected ten string metrics to evaluate performance. These metrics are as follows:

- Information Distance;
- Jaccard;
- Jaro;
- Jaro-Winkler;
- Levenshtein;
- Monge-Elkan;
- Needleman-Wunsch;
- Smith-Waterman;
- Soft TFIDF;
- TFIDF.

This selection was based on the popularity of the techniques in the literature as well as their relevance to this research.

5.2.1 Average Precision for Duplicate Detection Experiments

In this section, the evaluation results of the different methods used in duplicate detection tasks are described for the Life and Social Sciences datasets. The methods proposed in this work achieve the best values of average precision on three datasets out of five. On the remaining two datasets, the new methods produce top results approaching those of the best performers.

For the Animals dataset, the three best methods with the same values of average precision (0.95) are Jaccard, TFIDF and Soft TFIDF. They are followed by the three methods proposed in this work: HDTFJ with 0.93, HDLCP with 0.83, and HDSW with 0.64. The rest of the techniques demonstrated numbers close to zero.

For the Birds dataset, the best values of average precision (0.81) belong to TFIDF and Soft TFIDF. The next best results of 0.75 are of the HDTFJ and Jaro methods. The rest of the functions demonstrate similar numbers, except for the HDULCP and Information Distance methods, which are worse.

The best average precision of 0.94 on the Census dataset is achieved using the HDSW metric. It is followed by Smith-Waterman with 0.92, and Levenshtein and Needleman-Wunsch both measuring 0.90.

Table 5.2 Average Precision for Duplicate Detection Experiments

Metric	Dataset				
	Animals	**Birds**	**Census**	**Parks**	**Restaurants**
Information Distance	0.01	0.13	0.05	0.05	0.00
Jaccard	0.95	0.73	0.40	0.87	0.98
Jaro	0.06	0.75	0.73	0.94	0.92
Jaro-Winkler	0.04	0.74	0.71	0.95	0.93
Levenshtein	0.05	0.74	0.90	0.87	0.71
Monge-Elkan	0.08	0.71	0.76	0.95	0.75
Needleman-Wunsch	0.05	0.74	0.90	0.87	0.71
Smith-Waterman	0.09	0.43	0.92	0.81	0.86
Soft TFIDF	0.95	0.81	0.38	0.96	0.99
TFIDF	0.95	0.81	0.38	0.96	0.99
HDULCP	0.13	0.10	0.65	0.64	1.00
HDLCP	0.83	0.72	0.82	0.96	0.99
HDSW	0.64	0.72	0.94	0.94	0.89
HDTFJ	0.93	0.75	0.74	0.95	0.98

On the Parks dataset, three methods obtain the same best value of average precision: HDLCP, TFIDF, and Soft TFIDF. The next closest value of 0.95 is

achieved by the HDTFJ, Jaro-Winkler, and Monge-Elkan methods. The third best value of 0.94 belongs to the HDSW and Jaro methods.

On the Restaurants dataset, the best possible value of 1.00 is reached by the HDULCP method. The second best number is 0.99, produced by the HDLCP, TFIDF, and Soft TFIDF methods. The next best result of 0.98 belongs to HDTFJ and Jaccard.

5.2.2 Maximum F_1 for Duplicate Detection Experiments

The four HD-based methods achieve the best values of the F_1 metric on four datasets: 0.90 for the HDTFJ method on the Animals dataset, 0.88 for the HDSW method on the Census data, 0.95 by the HDLCP method on the Parks dataset, and 0.99 by the HDULCP method on the Restaurant data. On the Birds dataset, the best number of 0.86 belongs to the Jaccard method, which is closely approached by the following methods developed in this research: HDTFJ with 0.85, HDLCP and HDSW with 0.84.

Table 5.3 Maximum F_1 for Duplicate Detection Experiments

Metric	Dataset				
	Animals	**Birds**	**Census**	**Parks**	**Restaurants**
Information Distance	0.04	0.29	0.10	0.14	0.01
Jaccard	0.90	0.86	0.57	0.88	0.94
Jaro	0.12	0.82	0.69	0.91	0.90
Jaro-Winkler	0.09	0.82	0.65	0.92	0.94
Levenshtein	0.08	0.82	0.83	0.88	0.72
Monge-Elkan	0.13	0.85	0.70	0.94	0.75
Needleman-Wunsch	0.08	0.82	0.83	0.88	0.72
Smith-Waterman	0.15	0.54	0.85	0.77	0.81
Soft TFIDF	0.90	0.84	0.52	0.94	0.95
TFIDF	0.90	0.84	0.52	0.94	0.95
HDULCP	0.31	0.29	0.65	0.85	0.99
HDLCP	0.79	0.84	0.75	0.95	0.96
HDSW	0.74	0.84	0.88	0.93	0.83
HDTFJ	0.90	0.85	0.73	0.94	0.95

5.2.3 Execution Time for Duplicate Detection Experiments

The two fastest algorithms are the Jaro method, which requires the shortest time on the Animals, Birds, and Parks datasets and the Jaccard method, which has the best timing for the Census and Restaurants datasets. The Monge-Elkan and Information Distance metrics are the slowest in most cases. The Levenshtein and Needleman-Wunsch metrics demonstrate similar numbers because they are, in fact, similar techniques.

Table 5.4 Execution Time in Seconds for Duplicate Detection Experiments

Metric	Dataset				
	Animals	**Birds**	**Census**	**Parks**	**Restaurants**
Information Distance	3.50	0.42	7.57	1.25	40.03
Jaccard	0.22	0.02	0.15	0.07	1.99
Jaro	0.18	0.01	0.44	0.05	2.87
Jaro-Winkler	0.96	0.06	2.44	0.45	2.92
Levenshtein	1.32	0.07	5.58	0.67	37.31
Monge-Elkan	3.36	0.16	11.85	1.10	87.64
Needleman-Wunsch	1.33	0.07	5.57	0.37	37.31
Smith-Waterman	1.50	0.07	5.87	1.40	39.30
Soft TFIDF	0.99	0.04	1.17	0.37	23.28
TFIDF	0.28	0.02	0.22	0.08	2.98
HDULCP	2.20	0.12	1.37	0.76	12.34
HDLCP	1.75	0.09	1.23	0.56	12.54
HDSW	3.24	0.16	2.50	0.86	60.97
HDTFJ	2.02	0.11	1.44	0.62	15.68

5.2.4 Precision-Recall Curves for Duplicate Detection Experiments

The figures depicting precision-recall curves for duplicate detection experiments in the life and social sciences domain are shown in Appendix A. These charts correlate well with the average precision data shown in Table 5.2 but provide a more detailed outlook. Each of the figures in Appendix A depicts a precision-recall curve for one of the fourteen string metrics applied to one of the five datasets.

The precision-recall curves are an easy visual way to estimate an algorithm's performance. The general way to interpret a precision-recall chart is based on the location of a curve. When the curve goes through the lower-left section of the graph, then the method's overall performance is poor. In contrast, when the curve goes though the top right portion of the chart, the method's overall performance is good. On each of the charts, the horizontal axis shows the recall points from 0.0 to 1.0 and the vertical axis measures the interpolated average precision values, also from 0.0 to 1.0.

Figures A.1 and A.2 in Appendix A distinctively show that five string metrics have the best performance on the Animals dataset. These are Jaccard, Soft TFIDF, TFIDF, HDTFJ, and HDLCP. The pattern of the HDSW indicates worse results compared to the five leaders. The rest of the methods are far behind.

Figures A.3 and A.4 (see Appendix A) depict the precision-recall curves of the fourteen string similarity metrics for the Birds dataset. The curves of the Information Distance and HDULCP methods indicate low performance. The Smith-Waterman chart reflects moderate results. On the other hand, the rest of the methods have similar curves which indicate above average performance.

The next set of charts, shown in Figures A.5 and A.6 in Appendix A, portray the evaluation on the Census dataset. It is easy to notice that the HDSW curve has the highest precision values at the greatest number of recall points. It is followed by the similar curves of the Levenshtein, Needleman-Wunsch, and Smith-Waterman methods. The worst precision-recall dynamics belongs to Information Distance, TFIDF, and Soft TFIDF. These three curves have a similar trajectory, although the Information Distance has the deeper decrease of precision at the 0.1 point of recall.

Figures A.7 and A.8 (see Appendix A) portray the precision-recall charts for the Parks dataset. The pattern of performance of the string similarity metrics changes again as the characteristics of the dataset change. The eight metrics demonstrating excellent results and almost identical curves are the Jaro, Jaro-Winkler, Monge-Elkan, Soft TFIDF, TFIDF, HDLCP, HDSW, and HDTFJ methods. For these metrics, precision drops at the 0.9 recall point. The Jaccard, Levenshtein, and Needleman-Wunsch curves show a similar pattern but their precision drops occur at the 0.8 recall value. The Smith-Waterman and HDULCP curves lie lower on the chart, indicating lower performance. The Information Distance shows the worst performance.

The evaluation of the various distance metrics on the Restaurants dataset is shown in Figures A.9 and A.10 in Appendix A. The best performance is demonstrated by the HDULCP method. Its curve goes through the highest values of the precision up to 0.9 of recall and then drops insignificantly. Several methods have similar curves, namely HDLCP, TFIDF, Soft TFIDF, HDTFJ, and Jaccard. The Information Distance curve shows the worst performance at all recall points.

5.3 Evaluation of Clustering on Life and Social Sciences Data Sources

This section presents the evaluation of the selected string similarity metrics on the life and social sciences datasets for the clustering tasks. Four methods developed in this dissertation are benchmarked against the previously introduced ten methods of other researchers. In each of the Animals, Census, Parks, and Restaurants datasets, one of the proposed methods demonstrates superior performance in terms of average precision and maximum F_1. On the Birds dataset, three proposed methods obtain superior values.

5.3.1 Average Precision for Clustering Experiments

The four HD-based methods achieve the best values of average precision on four datasets during the clustering experiments: 0.54 for the HDTFJ on the Animals dataset, 0.46 by the HDSW on the Census dataset, 0.93 by the HDLCP on the Parks dataset, and 0.98 by the HDULCP on the Restaurants dataset.

On the Birds dataset, the HDTFJ, HDSW, and HDLCP methods obtain 0.71, 0.69, and 0.69 values of average precision respectively. These values are very close to the best value of 0.72 achieved by Jaro, Jaro-Winkler, TFIDF and Soft TFIDF. Other methods with superior performance are Jaccard (0.54) on the Animals data; Monge-Elkan (0.91), TFIDF (0.90), Soft TFIDF (0.90), HDSW (0.90), and HDTFJ (0.90) on the Parks dataset; TFIDF (0.98), Soft TFIDF (0.98), HDLCP (0.97), and HDTFJ (0.96) on the Restaurant data.

Table 5.5 Average Precision for Clustering Experiments

Metric	Dataset				
	Animals	**Birds**	**Census**	**Parks**	**Restaurants**
Information Distance	0.01	0.02	0.02	0.02	0.00
Jaccard	0.54	0.71	0.12	0.82	0.94
Jaro	0.23	0.72	0.26	0.89	0.76
Jaro-Winkler	0.23	0.72	0.26	0.89	0.89
Levenshtein	0.24	0.71	0.37	0.81	0.54
Monge-Elkan	0.07	0.69	0.27	0.91	0.49
Needleman-Wunsch	0.24	0.71	0.37	0.81	0.54
Smith-Waterman	0.25	0.19	0.38	0.61	0.32
Soft TFIDF	0.37	0.72	0.11	0.90	0.98
TFIDF	0.37	0.72	0.11	0.90	0.98
HDULCP	0.01	0.02	0.24	0.48	0.98
HDLCP	0.42	0.69	0.24	0.93	0.97
HDSW	0.26	0.69	0.46	0.90	0.71
HDTFJ	0.54	0.71	0.23	0.90	0.96

5.3.2 Maximum F_1 for Clustering Experiments

The four HD-based methods attain the best values of the maximum F_1 metric on four datasets during the clustering experiments: the HDTFJ method (0.66) on the Animals dataset, the HDSW method (0.65) on the Census dataset, the HDLCP method (0.92) on the Parks dataset, and the HDULCP method (0.95) on the Restaurants dataset. On the Birds dataset, the HDLCP and HDSW methods achieve a maximum F_1 value of 0.84, which is the second best result after the Monge-Elkan method (0.85). The HDTFJ result on the Birds data is 0.83.

Other superior results are achieved as follows: Jaccard (0.66) on the Animals dataset; Jaccard, TFIDF, and Soft TFIDF register 0.84 each on the Birds dataset; HDTFJ, TFIDF, and Soft TFIDF achieve 0.91 each on the Parks data, while HDSW obtains 0.90; TFIDF along with Soft TFIDF achieve 0.94 of maximum F_1 on the Restaurant data, followed by HDLCP (0.93) and HDTFJ (0.92).

Table 5.6 Maximum F_1 for Clustering Experiments

Metric	Dataset				
	Animals	**Birds**	**Census**	**Parks**	**Restaurants**
Information Distance	0.03	0.05	0.06	0.08	0.01
Jaccard	0.66	0.84	0.28	0.86	0.90
Jaro	0.36	0.81	0.41	0.88	0.73
Jaro-Winkler	0.36	0.80	0.41	0.88	0.88
Levenshtein	0.37	0.80	0.56	0.86	0.61
Monge-Elkan	0.28	0.85	0.44	0.89	0.53
Needleman-Wunsch	0.37	0.80	0.56	0.86	0.61
Smith-Waterman	0.35	0.32	0.58	0.62	0.43
Soft TFIDF	0.53	0.84	0.26	0.91	0.94
TFIDF	0.53	0.84	0.26	0.91	0.94
HDULCP	0.04	0.07	0.32	0.72	0.95
HDLCP	0.61	0.84	0.37	0.92	0.93
HDSW	0.37	0.84	0.65	0.90	0.70
HDTFJ	0.66	0.83	0.36	0.91	0.92

5.3.3 Execution Time for Clustering Experiments

The fastest methods on all the five life and social sciences datasets during clustering experiments are the Jaro and Jaro-Winkler methods, followed by Jaccard and TFIDF. The Information Distance and Monge-Elkan turn out to be the slowest similarity metrics for this type of experiment. The most significant difference in execution time appears on the Restaurant dataset evaluation. This dataset does not have the highest number of records, compared to the other evaluated data sources, but its average record length is the longest. This characteristic explains the observation of an increased execution time.

Table 5.7 Execution Time in Seconds for Clustering Experiments

Metric	Dataset				
	Animals	**Birds**	**Census**	**Parks**	**Restaurants**
Information Distance	63.53	6.57	12.51	11.29	96.00
Jaccard	1.39	0.11	0.45	0.20	6.28
Jaro	0.91	0.08	1.20	0.13	8.25
Jaro-Winkler	0.91	0.08	1.19	0.13	8.27
Levenshtein	6.40	0.55	13.31	0.92	93.82
Monge-Elkan	31.78	2.68	51.76	4.49	450.17
Needleman-Wunsch	6.33	0.54	13.26	0.92	93.17
Smith-Waterman	7.25	0.62	14.87	1.06	103.80
Soft TFIDF	5.42	0.48	3.25	1.00	107.12
TFIDF	2.10	0.19	0.77	0.37	11.31
HDULCP	6.98	0.58	2.12	0.94	22.60
HDLCP	6.18	0.57	2.13	0.93	23.05
HDSW	11.53	0.95	4.22	1.54	111.02
HDTFJ	7.15	0.63	2.48	1.11	28.41

5.3.4 Precision-Recall Curves for Clustering Experiments

The figures with precision-recall curves for clustering experiments in the life and social sciences domain are shown in Appendix B. The interesting fact is that most of the similarity metrics have similar shapes of precision-recall curves on each particular

dataset, varying by the degree of steepness and the recall point at which a precision drop occurs. This can be explained by the varying composition and characteristics of each particular dataset.

Figures B.1 and B.2 in Appendix B show the precision-recall charts for the clustering experiments on the Animals dataset. Except for the Information Distance and the HDULCP methods, which have the worst trajectories, other similarity functions have curve shapes indicating moderate performance. The Smith-Waterman and Monge-Elkan curves are smoother and the precision drops occur earlier, resulting in moderate performances. The Jaccard and HDTFJ curves have better shapes implying a better performance compared to the rest of the metrics.

In the Figures B.3 and B.4, the precision-recall charts for the Birds dataset are shown. Except for two outliers, Information Distance and HDULCP, the rest of the precision-recall curves represent very similar performances for all of the methods.

Again, in Figures B.5 and B.6 (Appendix B), precision-recall curves follow a similar pattern. Nevertheless, the HDSW method gets the highest precision values at most of the recall points when compared pairwise to every other method. This fact correlates well with the average precision data in Section 5.3.1. The HDULCP method starts with precision 1.0 at recall point 0.0, but its average precision is the same as that of the HDLCP method.

The Jaro, Jaro-Winkler, Monge-Elkan, Soft TFIDF, TFIDF, HDLCP, HDSW, and HDTFJ methods have similar trajectories in their precision-recall curves, as depicted in Figures B.7 and B.8 in Appendix B. These methods are followed by a second group consisting of the Jaccard, Levenshtein, and Needleman-Wunsch methods. Both the Smith-Waterman and HDULCP curves show a moderately good performance, though it is not clear from the charts, which of these methods has a

better curve. The average precision values given in Table 5.5 clearly demonstrate this case: the Smith-Waterman method (0.61) outperforms the HDULCP (0.48) method. The Information Distance method offers an inferior performance on this dataset as well.

Figures B.9 and B.10 clearly demonstrate the superior precision-recall performances for the HDULCP, TFIDF, and Soft TFIDF methods. It is easy to see that the next best values belong to the HDLCP and HDTFJ methods.

5.4 Evaluation of Duplicate Detection on Bioinformatics Data Sources

This section represents the results of the application of string similarity metrics to bioinformatics data for the duplicate detection tasks. The SPED method is benchmarked against ten selected similarity functions. The SPED method achieves the best possible results in terms of average precision and maximum F_1. The SPED method outperforms all the selected methods in maximum F_1 on all datasets and in average precision on three out of five datasets.

5.4.1 Average Precision for Duplicate Detection Experiments

This section presents experimental results expressed in average precision measures of the selected string similarity functions in the bioinformatics domain for the duplicate detection task. The SPED method, originally introduced in the authors' previous work [96, 123, 124], achieves the highest possible values on all five datasets and outperforms the rest of the benchmarked methods on three datasets.

The SPED, Soft TFIDF, and TFIDF methods achieve the maximum possible 1.0 value of average precision on the Buchnera Aphidicola Cedri Cinara and Paremecium Tetraurelia datasets. On the Bacteriophage T4, Carsonella Ruddii, and

Hyperthermus Butylicus, the SPED methods outperforms both the TFIDF and Soft TFIDF methods as well as the rest of the similarity metrics.

Table 5.8 Average Precision for Duplicate Detection Experiments

Metric	Dataset				
	Buchnera Aphidicola Cedri Cinara	**Bacteriophage T4**	**Carsonella Ruddii**	**Hyperthermus Butylicus**	**Paramecium Tetraurelia**
Information Distance	0.02	0.09	0.06	0.01	0.08
Jaccard	0.65	0.50	0.66	0.61	0.74
Jaro	0.28	0.41	0.31	0.26	0.28
Jaro-Winkler	0.37	0.67	0.32	0.32	0.31
Levenshtein	0.56	0.58	0.57	0.56	0.62
Monge-Elkan	0.54	0.55	0.57	0.55	0.62
MRFED	0.55	0.56	0.57	0.56	0.63
Needleman-Wunsch	0.56	0.58	0.57	0.56	0.62
Smith-Waterman	0.52	0.53	0.48	0.42	0.53
Soft TFIDF	1.00	0.97	0.99	0.99	1.00
TFIDF	1.00	0.97	0.99	0.99	1.00
SPED	1.00	1.00	1.00	1.00	1.00

The MRFED method gains better results than the Information Distance, Jaro and Smith-Waterman methods on all datasets. Also, it surpasses Jaro-Winkler and Monge-Elkan on most datasets.

5.4.2 Maximum F_1 for Duplicate Detection Experiments

Table 5.9 demonstrates that SPED surpasses the rest of the methods for duplicate detection. It is closely followed by the TFIDF and Soft TFIDF methods.

The MRFED method introduced in the authors' previous work [49, 50] demonstrates moderate results, outperforming the Information Distance, Jaro, and Smith-Waterman methods on all datasets. The MRFED method outperforms Jaro-Winkler on all the datasets, except for the Bacteriophage T4 dataset.

Table 5.9 Maximum F_1 for Duplicate Detection Experiments

Metric	Dataset				
	Buchnera Aphidicola Cedri Cinara	**Bacteriophage T4**	**Carsonella Ruddii**	**Hyperthermus Butylicus**	**Paramecium Tetraurelia**
Information Distance	0.06	0.17	0.13	0.03	0.16
Jaccard	0.67	0.62	0.72	0.70	0.79
Jaro	0.35	0.51	0.36	0.35	0.37
Jaro-Winkler	0.40	0.69	0.37	0.39	0.39
Levenshtein	0.62	0.63	0.65	0.66	0.67
Monge-Elkan	0.63	0.63	0.65	0.65	0.68
MRFED	0.62	0.62	0.64	0.65	0.69
Needleman-Wunsch	0.62	0.63	0.65	0.66	0.67
Smith-Waterman	0.57	0.51	0.57	0.45	0.59
Soft TFIDF	0.98	0.93	0.98	0.97	0.99
TFIDF	0.98	0.93	0.98	0.97	0.99
SPED	1.00	1.00	1.00	1.00	1.00

5.4.3 Execution Time for Duplicate Detection Experiments

The shortest execution time during the duplicate detection experiments on the bioinformatics datasets is demonstrated by the Jaccard method. The TFIDF time is second best. It differs from the Jaccard by a factor of approximately 2.0. It is worth noting that the SPED method improves the execution time of the author's previous work, the MRFED method. The longest execution time is exhibited by MRFED, the next result belongs to the Monge-Elkan, followed by the SPED, Smith-Waterman, and Levenshtein techniques.

Table 5.10 Execution Time for Duplicate Detection Experiments

Metric	Dataset				
	Buchnera Aphidicola Cedri Cinara	**Bacteriophage T4**	**Carsonella Ruddii**	**Hyperthermus Butylicus**	**Paramecium Tetraurelia**
Information Distance	517	22	45	812	51
Jaccard	240	5	26	465	32
Jaro	880	16	85	1582	115
Jaro-Winkler	880	16	86	1593	116
Levenshtein	12047	222	1196	21727	1663
Monge-Elkan	46591	856	4575	86964	6638
MRFED	73209	1470	7546	170669	13744
Needleman-Wunsch	9973	183	985	18293	1394
Smith-Waterman	12940	241	1303	23503	1775
Soft TFIDF	5585	96	525	9415	643
TFIDF	478	10	50	877	60
SPED	26388	483	2584	47890	3720

5.4.4 Precision-Recall Curves for Duplicate Detection Experiments

Appendix C contains precision-recall charts for the duplicate detection experiments on the bioinformatics datasets. Figures C.1, C.2, C.3, C.4, and C.5 display precision-recall charts for the twelve string similarity metrics selected for this type of experiments.

The methods exhibit almost identical curve trajectories, insignificantly varying from dataset to dataset. An interesting observation concerns SPED, TFIDF, and Soft TFIDF. On every chart, SPED gets 1.0 precision at every recall point. The TFIDF and Soft TFIDF methods experience a slight precision drop at the recall of 1.0. Still, this drop does not affect their average precision on the Buchnera Aphidicola Cedri Cinara and Paramecium Tetraurelia datasets, but decreases the precision on average for the rest of the data sources.

5.5 Evaluation of Clustering on Bioinformatics Data Sources

This section describes the results of the evaluation of string similarity functions for the clustering task applied to the bioinformatics data. The evaluation is done by taking the same approach of measuring the average precision, maximum F_1, and execution time, as well as plotting the precision-recall charts. This section presents the experimental results for the following methods: Information Distance, Jaccard, Jaro, Jaro-Winkler, Levenshtein, Monge-Elkan, MRFED, Needleman-Wunsch, Smith-Waterman, Soft MRFED, Soft TFIDF, TFIDF, and SPED.

Similar to the duplicate detection experiments presented in Section 5.4, the SPED method achieves the highest possible values for average precision and maximum F_1 on all the evaluated bioinformatics datasets. SPED outperforms the rest of the methods in terms of maximum F_1 on all the datasets. It surpasses other evaluated methods on the average precision measure for three datasets, whereas the TFIDF and Soft TFIDF methods have matching values of average precision on two datasets.

5.5.1 Average Precision for Clustering Experiments

The SPED method demonstrated the maximum possible value of 1.0 average precision on all the datasets in the clustering experiments. The TFIDF and Soft TFIDF methods achieved the best value of 1.0 on two datasets and showed top results on the other three data sources. The predecessor of the SPED method, MRFED, demonstrated moderate performance. Its average precision values lie in the [0.5, 0.6] interval.

Taking a closer look at the rest of the methods, Jaccard achieves a superior value for average precision on the Paramecium Tetraurelia, Carsonella Ruddii,

Hyperthermus Butylicus, and Buchnera Aphidicola Cedri Cinara datasets. The Jaro-Winkler method has the best average precision among the remaining set of metrics on the Bacteriophage T4.

Table 5.11 Average Precision for Clustering Experiments

Metric	Dataset				
	Buchnera Aphidicola Cedri Cinara	**Bacteriophage T4**	**Carsonella Ruddii**	**Hyperthermus Butylicus**	**Paramecium Tetraurelia**
Information Distance	0.02	0.07	0.05	0.01	0.06
Jaccard	0.64	0.47	0.65	0.59	0.73
Jaro	0.27	0.39	0.29	0.24	0.25
Jaro-Winkler	0.35	0.65	0.30	0.30	0.28
Levenshtein	0.54	0.55	0.55	0.55	0.59
Monge-Elkan	0.54	0.54	0.56	0.54	0.59
MRFED	0.54	0.53	0.52	0.50	0.61
Needleman-Wunsch	0.54	0.55	0.55	0.55	0.59
Smith-Waterman	0.51	0.51	0.46	0.39	0.51
Soft TFIDF	1.00	0.95	0.97	0.98	1.00
TFIDF	1.00	0.95	0.97	0.98	1.00
SPED	1.00	1.00	1.00	1.00	1.00

The MRFED method outperforms the Information Distance, Jaro, Jaro-Winkler, Levenshtein, Monge-Elkan, Needleman-Wunsch and Smith-Waterman methods on the Paramecium Tetraurelia dataset. The MRFED method has better results than the Information Distance, Jaccard, Jaro, and Smith-Waterman methods on the Bacteriophage T4. On the Carsonella Ruddii, Hyperthermus Butylicus, and Aphidicola Cedri Cinara datasets, MRFED achieves higher values of average precision than the Information Distance, Jaro, Jaro-Winkler, and Smith-Waterman methods. The comparison above shows that the MRFED has an average performance

on the Gene Ontology Annotation (GOA) datasets, though it outperforms several well-known string similarity metrics.

5.5.2 Maximum F_1 for Clustering Experiments

Table 5.12 demonstrates the maximum F_1 measure values for the string metrics evaluation performed on the GOA datasets for clustering tasks. Again, as in the case of the duplicate detection task for the bioinformatics datasets, SPED has the highest values on all the datasets. It outperforms all other evaluated metrics, though the TFIDF and Soft TFIDF methods show very close results.

Table 5.12 Maximum F_1 for Clustering Experiments

Metric	Dataset				
	Buchnera Aphidicola Cedri Cinara	**Bacteriophage T4**	**Carsonella Ruddii**	**Hyperthermus Butylicus**	**Paramecium Tetraurelia**
Information Distance	0.06	0.16	0.12	0.03	0.14
Jaccard	0.67	0.61	0.71	0.70	0.78
Jaro	0.33	0.49	0.34	0.33	0.34
Jaro-Winkler	0.39	0.66	0.35	0.37	0.36
Levenshtein	0.62	0.63	0.64	0.66	0.65
Monge-Elkan	0.63	0.63	0.64	0.65	0.66
MRFED	0.62	0.62	0.58	0.49	0.67
Needleman-Wunsch	0.62	0.63	0.64	0.66	0.65
Smith-Waterman	0.57	0.49	0.56	0.43	0.57
Soft TFIDF	0.98	0.91	0.93	0.95	0.98
TFIDF	0.98	0.91	0.93	0.95	0.98
SPED	1.00	1.00	1.00	1.00	1.00

Consider the remaining ten metrics as a separate set. The Jaccard metric achieved the best performance on four datasets: the Parametrium Tetraurelia, Carsonella Ruddii, Hyperthermus Butylicus, and Buchnera Aphidicola Cedri Cinara.

The best performance on the Bacteriophage T4 dataset was obtained by the Jaro-Winkler metric.

The MRFED outperformed the following similarity metrics:

- Information Distance, Jaro, Jaro-Winkler, Levenshtein, Monge-Elkan, Needleman-Wunsch, Smith-Waterman on the first dataset;
- Information Distance, Jaccard, Jaro, Smith-Waterman on the second dataset;
- Information Distance, Jaro, Jaro-Winkler, Smith-Waterman on the third, fourth, and fifth datasets.

The MRFED method is not the best string similarity metric for duplicate detection on the GOA datasets but it was able to outperform several well-known widely used methods.

5.5.3 Execution Time for Clustering Experiments

In terms of the execution time, the Jaccard method earned the best results on all datasets. The TFIDF metric had the second best running time on each of the bioinformatics datasets.

The MRFED method was the slowest on the first, second, third, and fifth datasets. Soft TFIDF showed the worst execution time on the Hyperthermus Butylicus dataset. The Levenshtein, Monge-Elkan, Needleman-Wunsch, and SPED methods were among the slowest.

It is worth noting that an execution time shown in Table 5.13 is not a perfect estimate of its method's run time, as it may be skewed by the available amount of operating memory, processor load, and other factors affecting the experiment's flow. These factors don't have any influence on the average precision, maximum F_1, and precision-recall charts. However, one of the best ways to compare algorithm run times remains the analysis of their computational complexities.

Table 5.13 Execution Time in Seconds for Clustering Experiments

Metric	Dataset				
	Buchnera Aphidicola Cedri Cinara	**Bacteriophage T4**	**Carsonella Ruddii**	**Hyperthermus Butylicus**	**Paramecium Tetraurelia**
Information Distance	1,054	91	408	3,361	249
Jaccard	512	7	38	1,071	45
Jaro	2,350	22	115	2,619	160
Jaro-Winkler	1,461	23	134	2,731	160
Levenshtein	101,068	302	1,848	34,789	2,670
Monge-Elkan	62,463	788	4,257	72,022	6,363
MRFED	134,595	8,529	6,565	6,969	23,036
Needleman-Wunsch	21,164	382	1,920	37,285	2,665
Smith-Waterman	21,346	361	1,847	33,759	2,622
Soft TFIDF	62,385	83	429	199,743	506
TFIDF	642	14	61	1,605	71
SPED	65,150	508	5,244	96,405	3,845

5.5.4 Precision-Recall Curves for Clustering Experiments

Appendix D shows precision-recall charts for the string distance evaluation of clustering tasks on the bioinformatics datasets. Each method shows a very similar precision-recall curve for each of the datasets, with only slight differences. This is explained by the similar dataset characteristics. In all cases, except for the Information Distance method in Figures D.1 and D.2, curves start at the upper left corner with the 1.0 value of precision. In almost all cases, except for SPED, TFIDF, and Soft TFIDF, the precision declines and reaches values close to zero at the 1.0 point of recall.

Nevertheless, the SPED, TFIDF, and Soft TFIDF methods demonstrate superior performance in Figures D.1 through D.5. SPED obtains 1.0 values of precision at each point of recall. In Figures D.1 and D.5, the TFIDF and Soft TFIDF methods manage to keep up a 1.0 level of precision, up to the 0.9 point of recall and

experience a precision drop at the last recall point. The TFIDF and Soft TFIDF methods have a declining precision starting at the first point of recall and continue the decline up to the last point in Figures D.2, D.3, and D.4. This behavior is reflected in the average precision values given in Table 5.11 of Section 5.5.1.

5.6 Evaluation of Duplicate Detection on Medical Informatics Data Sources

The results of the string metrics evaluation for duplicate detections tasks on the medical informatics datasets are presented in this section. Three methods proposed by the author, HDLCP, HDTFJ, and LACP, are benchmarked against ten popular string similarity metrics. On each of the datasets, the new methods outperform the rest of the evaluated methods in the maximum F_1 measure. In average precision, the new methods surpass the rest of the techniques on the second and third datasets, achieve superior results on the first data source, and are among the top performers on the forth dataset.

5.6.1 Average Precision for Duplicate Detection Experiments

This section shows that the new methods created in this research demonstrate superior values of average precision for duplicate detection tasks on the medical informatics datasets. For the UMLS Longest Concepts dataset, the HDLCP and HDTFJ methods, along with TFIDF and Soft TFIDF show a 0.25 average precision, which is the best value for this dataset. On the SNOMED Longest Concepts and the UMLS Most Frequent Concepts datasets, the LACP method excels over the other methods with 0.84 and 0.62 values of average precision, respectively. On the SNOMED Longest Concepts dataset, HDLCP and HDTFJ show the second best average precision value of 0.72. For the SNOMED Most Frequent Concepts dataset, the TFIDF and Soft

TFIDF methods demonstrate the best average precision of 0.55. The closest following results for this dataset are shown by the HDLCP (0.52) and LACP (0.51) methods.

Table 5.14 Average Precision for Duplicate Detection Experiments

Metric	Dataset			
	UMLS Longest Concepts	SNOMED Longest Concepts	UMLS Most Frequent Concepts	SNOMED Most Frequent Concepts
Information Distance	0.00	0.00	0.04	0.03
Jaccard	0.22	0.54	0.31	0.33
Jaro	0.14	0.69	0.26	0.40
Jaro-Winkler	0.14	0.69	0.44	0.45
Levenshtein	0.18	0.54	0.16	0.21
Monge-Elkan	0.12	0.65	0.22	0.32
Needleman-Wunsch	0.18	0.54	0.16	0.21
Smith-Waterman	0.09	0.34	0.18	0.16
Soft TFIDF	0.25	0.69	0.51	0.55
TFIDF	0.25	0.69	0.51	0.55
HDLCP	0.25	0.72	0.50	0.52
HDTFJ	0.25	0.72	0.36	0.42
LACP	0.12	0.84	0.62	0.51

5.6.2 Maximum F_1 for Duplicate Detection Experiments

The new methods demonstrate the highest values of maximum F_1 on each medical informatics dataset for duplicate detection experiments. On the UMLS Longest Concepts dataset, the HDLCP and HDTFJ methods produce a maximum F_1 value of 0.41. For the SNOMED Longest Concepts, UMLS Most Frequent Concepts, and SNOMED Most Frequent Concepts data, the LACP achieves the highest values of maximum F_1: 0.92, 0.69, and 0.61, respectively.

Table 5.15 Maximum F_1 for Duplicate Detection Experiments

Metric	Dataset			
	UMLS Longest Concepts	SNOMED Longest Concepts	UMLS Most Frequent Concepts	SNOMED Most Frequent Concepts
Information Distance	0.03	0.02	0.07	0.07
Jaccard	0.37	0.59	0.33	0.38
Jaro	0.28	0.77	0.33	0.49
Jaro-Winkler	0.28	0.77	0.56	0.57
Levenshtein	0.33	0.65	0.21	0.28
Monge-Elkan	0.26	0.67	0.24	0.37
Needleman-Wunsch	0.33	0.65	0.21	0.28
Smith-Waterman	0.18	0.38	0.21	0.22
Soft TFIDF	0.40	0.70	0.49	0.58
TFIDF	0.40	0.70	0.49	0.58
HDLCP	0.41	0.73	0.48	0.57
HDTFJ	0.41	0.74	0.36	0.46
LACP	0.27	0.92	0.69	0.61

5.6.3 Execution Time for Duplicate Detection Experiments

One of the newly developed methods, LACP, demonstrates the shortest execution time for all medical informatics datasets. This fact, in combination with the superior values of average precision and maximum F_1, shows it to be the best method for duplicate detection tasks according to the performed evaluation. The LACP method shows the shortest execution time (in seconds) on each of the datasets, namely 202, 40, 11, and 202 for the datasets one through four, respectively.

Table 5.16 Execution Time in Seconds for Duplicate Detection Experiments

Metric	Dataset			
	UMLS Longest Concepts	SNOMED Longest Concepts	UMLS Most Frequent Concepts	SNOMED Most Frequent Concepts
Information Distance	1022	949	396	1022
Jaccard	568	70	20	568
Jaro	3637	105	25	3637
Jaro-Winkler	3617	115	26	3617
Levenshtein	57811	1273	301	57811
Monge-Elkan	258502	6240	1340	258502
Needleman-Wunsch	57982	1294	258	57982
Smith-Waterman	58753	1444	293	58753
Soft TFIDF	16874	806	174	16874
TFIDF	928	132	37	928
HDLCP	2364	247	67	2364
HDTFJ	1947	301	82	1947
LACP	202	40	11	202

5.6.4 Precision-Recall Curves for Duplicate Detection Experiments

Figures E.1 through E.4 illustrate precision-recall dependencies of the duplicate detection evaluation on medical informatics data. Each method curve follows approximately the same pattern from dataset to dataset. The pattern differences are in the degree of precision drop and the recall point where this drop occurs. The LACP method shows higher precision values at more recall points on most figures.

5.6.5 LACP-Based Interactive Spell Checker

As one more way to evaluate the LACP method's performance, this research demonstrates an interactive online spell checker [126] developed by the author. It is based on the LACP method and checks the spelling of SNOMED CT terms. The spell checker is a program written in the PHP language, which connects to a MySQL database containing SNOMED CT terms from the 2009AB edition of the UMLS. The

goal of the application is to evaluate the LACP performance, so that the similarity of resulting sets may be estimated at a glance. It also provides a practical application that can be perused by non-expert end users of SNOMED CT.

The LACP spell checker accepts an input query and interactively outputs the SNOMED CT terms satisfying the condition $LACP(S, T) < threshold$. Here, S is an input query, and T is a SNOMED CT term. To reduce the running time, the algorithm limits the set of search terms applying length criteria as described below.

Three modes of operation were implemented: (a) search with dynamically estimated parameters; (b) search with static parameters; and (c) search with user-defined parameters. In case (a), the search is limited to the database terms meeting the criterion (5.1); α is defined in (5.2), *threshold* is 0.1.

$$\max(0,|S|-\left\lceil\frac{|S|}{10}\right\rceil-3)<|T|<|S|+\left\lceil\frac{|S|}{10}\right\rceil+3 \tag{5.1}$$

In case (a), the parameter α is set individually for each pair of strings S and T as shown in (5.2):

$$\alpha=\left\lceil\frac{\min(|S|,|T|)}{5}\right\rceil \tag{5.2}$$

In case (b), α is set to 1, *threshold* is 0.1, and the length of a term should be in the following range:

$$\max(0,|S|-3)<|T|<|S|+3 \tag{5.3}$$

In case (c), a user selects values of the parameters from the predefined sets. The search is restricted to terms with lengths in the interval (5.4).

$$\max(0,|S|-a)<|T|<|S|+b \tag{5.4}$$

Parameters a, b, and α are constrained to integers in the interval 1..15, and *threshold* should be selected from the set (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0).

The spell checker returns the following information for each matching term T: a sequence number; the value *LACP*(*S*,*T*); the string representation of the similar term T; the CUI of the term; and the value of α.

Table 5.17 depicts results returned for the input term "Ischemia" (case (a)); static search for "Haemophilia", (case (b)); and user-defined search for "Ammonium" with the following parameter values $a = 3$, $b = 3$, *threshold* = 0.1 (case (c)). The example in Table 5.17 shows that the spell checker returns closely related matches for the input queries.

Table 5.17 Results Returned by the Spell Checker

Case (a): Dynamic search for "Ischemia"

#	LACP Value	SNOMED CT Term	CUI	α
1	0	Ischemia	C0022116	2
2	0	Ischemic	C0475224	2
3	0.06	Ischaemia	C0022116	2
4	0.06	Ischaemic	C0475224	2

Case (b): Static search for "Haemophilia"

#	LACP Value	SNOMED CT Term	CUI	α
1	0	Haemophilia	C0684275	1
2	0	Haemophilia	C1321589	1
3	0.08	Haemophilia B	C0008533	1
4	0.08	Haemophilia C	C0015523	1
5	0.08	Haemophilia A	C0019069	1

Case (c): User-defined search for "Ammonium", a = 3, b = 3, *threshold* = 0.1

#	LACP Value	SNOMED CT Term	CUI	α
1	0	Ammonium	C0002611	2
2	0.07	Ammonia	C0002607	2

The examples below demonstrate the dynamic search for a SNOMED CT term with intentionally introduced misspellings. The spell checker returns the correct term Vertebrate, when a single misspelling is introduced (Table 5.18, case (a)). There are four records returned with the correct term among them, when two characters are wrong (Table 5.18, case (b)).

Table 5.18 Search Results for the Misspelled Term "Vertebrate"

Case (a): Dynamic search for "Verte**p**rate"

#	LACP Value	SNOMED CT Term	CUI	α
1	0	Vertebrate	C0042567	2

Case (b): Dynamic search for "Verte**p**rate**e**"

#	LACP Value	SNOMED CT Term	CUI	α
1	0	Temperature	C0039476	3
2	0	Overtreated	C1273485	3
3	0.05	Vertebrate	C0042567	2
4	0.08	Perseveration	C0233651	3

The next example in Table 5.19 depicts the results of the dynamic search for the misspelled term Sodium Fluoride. In case (a), the search phrase has one misspelled character and one missing character. The correct term is returned along with three other terms. In case (b), the number of incorrect term is decreased by one, while the number of misspellings is increased to two.

Table 5.19 Search Results for the Misspelled Term "Sodium Fluoride"

Case (a): Dynamic search for "Sod**e**um floride**"**

#	LACP Value	SNOMED CT Term	CUI	α
1	0.03	Sodium fluoride	C0037508	3
2	0.03	Sodium feredate	C0357084	3
3	0.04	Sodium folate	C0304894	3
4	0.1	Sodium feredetate	C0357084	3

Case (b): Dynamic search for "Sod**e**um Florid**a**"

#	LACP Value	SNOMED CT Term	CUI	α
1	0.03	Sodium fluoride	C0037508	3
2	0.03	Sodium feredate	C0357084	3
3	0.04	Sodium folate	C0304894	3

The search for the misspelled term Pancreatitis is shown in Table 5.20. In both cases (a) and (b), two misspellings are introduced in the search phrases. The spell checker returns only the correct term in case (a). In case (b), the term Turkmenistan is returned as well as the correct term.

Table 5.20 Search Results for the Misspelled Term "Pancreatitis"

Case (a): Dynamic search for "**B**an**k**reatitis"

#	LACP Value	SNOMED CT Term	CUI	α
1	0	Pancreatitis	C0030305	3

Case (b): Dynamic search for "**P**un**k**reatitis"

#	LACP Value	SNOMED CT Term	CUI	α
1	0	Pancreatitis	C0030305	3
2	0	Turkmenistan	C0041403	3

The results provided by the dynamic search are positive, returning the related terms. Still, unrelated terms may be displayed within the search results. The spell checker allows to set the parameter values manually to adjust the performance. It is

possible to include or remove records from a resulting set by tuning the parameters in the Search in the User-Defined Parameters mode of the spell-checker.

5.7 Evaluation of Clustering on Medical Informatics Data Sources

The section presents an evaluation for clustering tasks of the selected string similarity metrics on medical informatics data. Similar to the duplicate detection on the medical informatics data shown in the Section 5.6, the proposed methods demonstrate superior results in the maximum F_1 measure on every dataset and superior values of average precision on three out of four data sources.

5.7.1 Average Precision for Clustering Experiments

The three proposed methods achieve top performance in average precision on all datasets and the best results on three out of four data sources. The HDLCP and HDTFJ methods obtain a value of 0.25 as average precision on the UMLS Longest Concepts data. The same number is produced by the TFIDF and Soft TFIDF methods. The LACP method gets the highest scores on the SNOMED Longest Concepts data (0.84) and on the UMLS Most Frequent Concepts (0.62). The HDLCP and HDTFJ methods have the next best value of 0.72 average precision on the SNOMED Longest Concepts data source. The TFIDF and Soft TFIDF methods demonstrate the best average precision of 0.55 on the SNOMED Most Frequent Concepts dataset. The closest results of 0.52 and 0.51 belong to the HDLCP and LACP methods.

Table 5.21 Average Precision for Clustering Experiments

Metric	Dataset			
	UMLS Longest Concepts	SNOMED Longest Concepts	UMLS Most Frequent Concepts	SNOMED Most Frequent Concepts
Information Distance	0.00	0.00	0.03	0.03
Jaccard	0.21	0.46	0.30	0.31
Jaro	0.14	0.69	0.25	0.39
Jaro-Winkler	0.13	0.69	0.43	0.44
Levenshtein	0.17	0.53	0.15	0.20
Monge-Elkan	0.14	0.55	0.20	0.29
Needleman-Wunsch	0.17	0.54	0.15	0.20
Smith-Waterman	0.08	0.33	0.18	0.15
Soft TFIDF	0.24	0.42	0.50	0.53
TFIDF	0.24	0.42	0.50	0.53
HDLCP	0.21	0.75	0.49	0.50
HDTFJ	0.24	0.58	0.35	0.40
LACP	0.11	0.85	0.61	0.50

5.7.2 Maximum F_1 for Clustering Experiments

The proposed methods outrank the benchmarked ones on the maximum F_1 measure on every evaluated dataset. On the first dataset, the highest score of 0.40 belongs to the HDTFJ method. The best scores for the second, third, and fourth data sources are achieved by the LACP method: 0.92, 0.69, and 0.60, respectively.

Table 5.22 Maximum F_1 for Clustering Experiments

Metric	Dataset			
	UMLS Longest Concepts	SNOMED Longest Concepts	UMLS Most Frequent Concepts	SNOMED Most Frequent Concepts
Information Distance	0.02	0.01	0.07	0.07
Jaccard	0.36	0.56	0.33	0.37
Jaro	0.27	0.77	0.32	0.48
Jaro-Winkler	0.27	0.77	0.55	0.56
Levenshtein	0.32	0.64	0.21	0.27
Monge-Elkan	0.30	0.65	0.22	0.35
Needleman-Wunsch	0.32	0.65	0.21	0.27
Smith-Waterman	0.17	0.37	0.20	0.21
Soft TFIDF	0.39	0.45	0.48	0.57
TFIDF	0.39	0.45	0.48	0.57
HDLCP	0.38	0.74	0.47	0.56
HDTFJ	0.40	0.66	0.36	0.45
LACP	0.25	0.92	0.69	0.60

5.7.3 Execution Time for Clustering Experiments

The best method in terms of the execution time for clustering experiments on the medical informatics datasets is LACP. The LACP method demonstrated the shortest time (in seconds) on each of the datasets: 644, 480, 68, and 13 for the first through fourth datasets, respectively.

Considering results for all three measures shown in the Sections 5.7.1 through 5.7.3, LACP turns out to be the best performing method for the clustering tasks on the evaluated medical informatics datasets. The other methods introduce in this research also show top performance.

Table 5.23 Execution Time in Seconds for Clustering Experiments

Metric	Dataset			
	UMLS Longest Concepts	SNOMED Longest Concepts	UMLS Most Frequent Concepts	SNOMED Most Frequent Concepts
Information Distance	1326	1102	2555	287
Jaccard	12347	544	119	166
Jaro	6144	1990	221	39
Jaro-Winkler	6315	2288	203	39
Levenshtein	97833	28069	2099	442
Monge-Elkan	442106	137657	21885	2174
Needleman-Wunsch	97820	15886	2131	427
Smith-Waterman	101092	31134	2431	579
Soft TFIDF	369720	143353	1342	291
TFIDF	20060	1006	221	110
HDLCP	3139	2201	495	136
HDTFJ	4771	2814	605	165
LACP	644	480	68	13

5.7.4 Precision-Recall Curves for Clustering Experiments

Figures F.1 through F.4 illustrate precision-recall curves for the thirteen evaluated similarity functions on the medical informatics datasets for clustering tasks. Analysis of the charts shows that most curves start at the top value of precision at the 0.0 recall point and then drop to the lowest value. The better performing similarity metrics stay at the higher values of precision longer and experience the precision decrease later, compared to the worst performing metrics. The LACP method demonstrates the distinctively superior curve trajectories in Figures F.1 and F.4. In Figures F.2 and F.3, the TFIDF, Soft TFIDF, HDLCP, HDTFJ, and LACP curves lie close to one another, indicating similar performances.

CHAPTER 6

SUMMARY

6.1 Discussion

Several new approximate string matching methods, designed in this research, were evaluated for duplicate detection and clustering tasks on datasets from the life and social sciences, bioinformatics, and medical informatics domains. These methods were benchmarked against ten well-known and widely used string similarity metrics.

The MRFED and SPED methods developed in this research were evaluated on the bioinformatics datasets. The initially developed MRFED method suffered from moderate performance and long execution times. The SPED method was developed to overcome these problems, improve results and decrease run times. The SPED computational complexity is $O(n^2)$, which is the same as that of the MRFED method. The performance evaluation experiments showed a decrease in execution time by a constant factor. Also, the SPED method significantly improved the performance of the MRFED method.

The SPED method was described in previously published work of the author. An evaluation of the utility of a different SPED version, without the use of the re-scorer, on the bioinformatics domain has been presented in previously published work [123, 124]. The SPED method with the implementation of the Winkler-like re-scorer has been applied to the medical informatics domain in the past [96]. In this thesis, the SPED method with the re-scorer was applied to the bioinformatics domain for the first time. It showed outstanding results in both duplicate detection and clustering experiments. The SPED achieved the highest possible values of average precision and maximum F_1 measures on all datasets used in the evaluation.

The new HD method and its four modifications were proposed in Section 4.3. These methods targeted the social and life sciences domain. This family of methods outperformed ten well-known similarity methods used on four datasets in terms of average precision and maximum F_1. Excellent results were obtained on the remaining data sources. These positive outcomes were reached for both, duplicate detection and clustering tasks.

The LACP method is presented in this dissertation for the first time. It was designed for duplicate detection and clustering tasks in the medical informatics domain. Also, the HDLCP and HDTFJ methods were chosen for evaluation on medical informatics datasets. The HDCLP and HDTFJ methods showed the best average precision along with TFIDF and Soft TFIDF on the UMLS Longest Concepts dataset. The HDTFJ method achieved the best values of maximum F_1 on the UMLS Longest Concepts dataset. These successful results were obtained for both duplicate detection and clustering tasks.

The LACP metric outperformed the rest of the benchmarked methods in average precision on two out of four datasets. It showed the best numbers of maximum F_1 on three out of four data sources. Also, the LACP method showed the shortest computational time on all bioinformatics datasets.

6.2 Conclusions

This work demonstrates the effectiveness of several new string similarity metrics for duplicate detection and clustering tasks in the social and life sciences, bioinformatics, and medical informatics domains. These methods show superior, and in certain cases outstanding results compared to ten well-known and widely used similarity functions.

Based on the experimental results obtained during this research, the author concludes: SPED achieves the best results applied to the bioinformatics datasets, LACP shows outstanding performance on the medical informatics datasets, and the HD-based methods demonstrate superior results on the life and social sciences datasets. The MRFED method, developed as the starting point of this research, does not produce competitive results.

Described in detail, the motivation and research problems support the importance of this research in the domains of social sciences, bioinformatics and medical informatics. Extensive evaluations were performed, to validate the proposed methods. In the majority of experiments, the new methods introduced by the author in this work achieved the highest values of the measures used for performance evaluation. Two outcomes deserve particular attention: (1) the SPED method gained the highest possible values of average precision and maximum F_1 on bioinformatics datasets in all performed experiments; (2) the LACP method produced the best results on three out of four medical informatics datasets in the shortest time with the lowest computational complexity of $O(n)$.

6.3 Future Work

The main direction of future work is to combine the string similarity metrics developed by the author into one compound method. This new method should produce an automated decision indicating which inner method is most appropriate for a particular case. This decision could be based on the evaluation of a subset of a dataset. Tan et al. [125] empirically show that a universal method with superior performance in all domains does not exist. It was also shown that the performance of metrics varies for datasets from different domains. Thus, a compound method as

described above would make a choice of which method to use, instead of using one specialized method. Such a combined method would save labor and time of terminology integrators.

Another direction for future work lies in the modification of the SPED method. A new formula for histogram difference was introduced in Section 4.3. It is possible to use this formula to calculate the node values in connection with the SPED algorithm. Further evaluation of this technique is necessary to test the validity of this proposed approach.

APPENDIX A

PRECISION-RECALL CHARTS FOR DUPLICATE DETECTION EXPERIMENTS ON LIFE AND SOCIAL SCIENCES DATASETS

This appendix provides precision-recall charts for the evaluation presented in the Section 5.2.4.

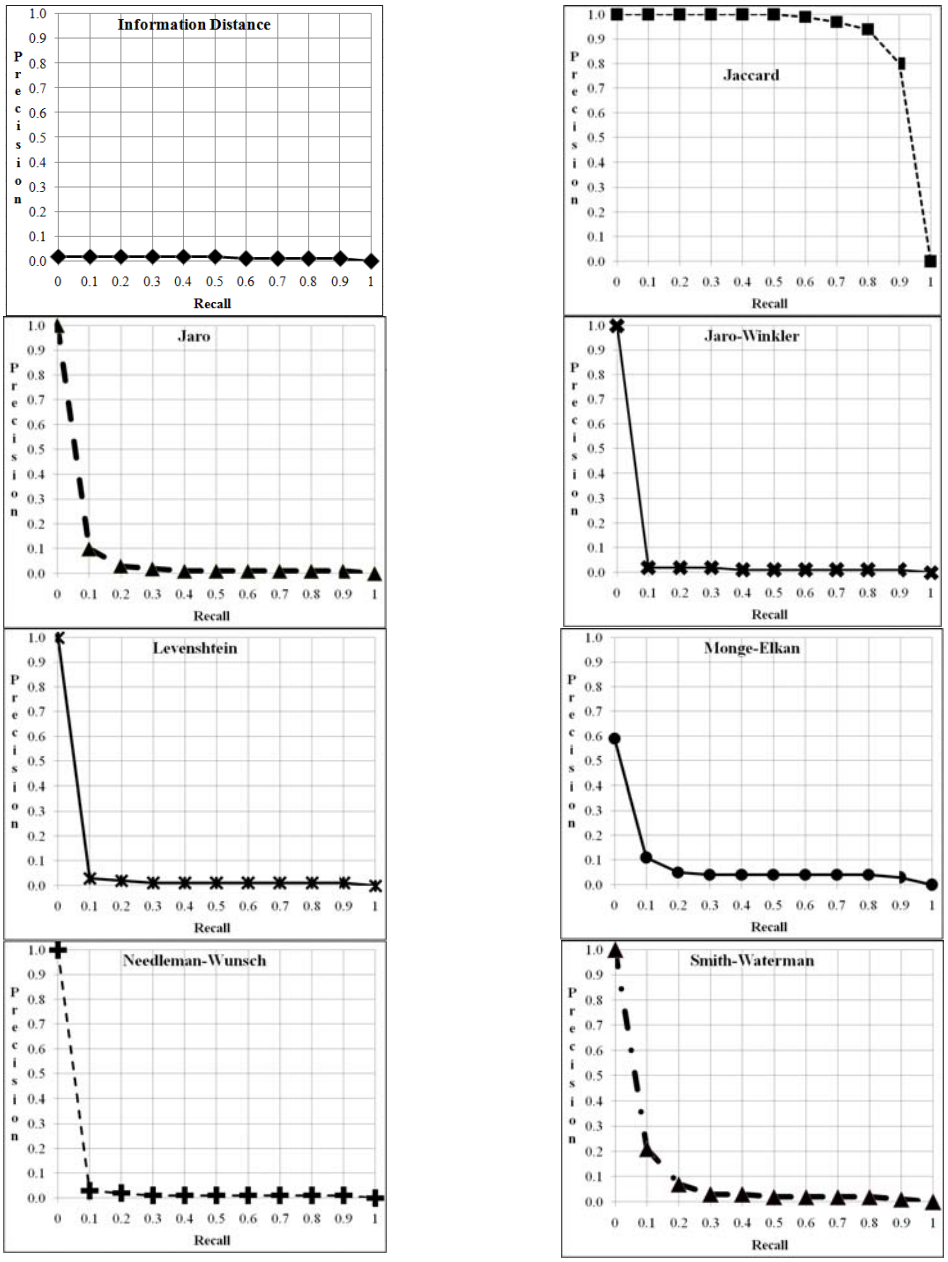


Figure A.1 Precision-recall curves for the duplicate detection experiments on the Animals dataset.

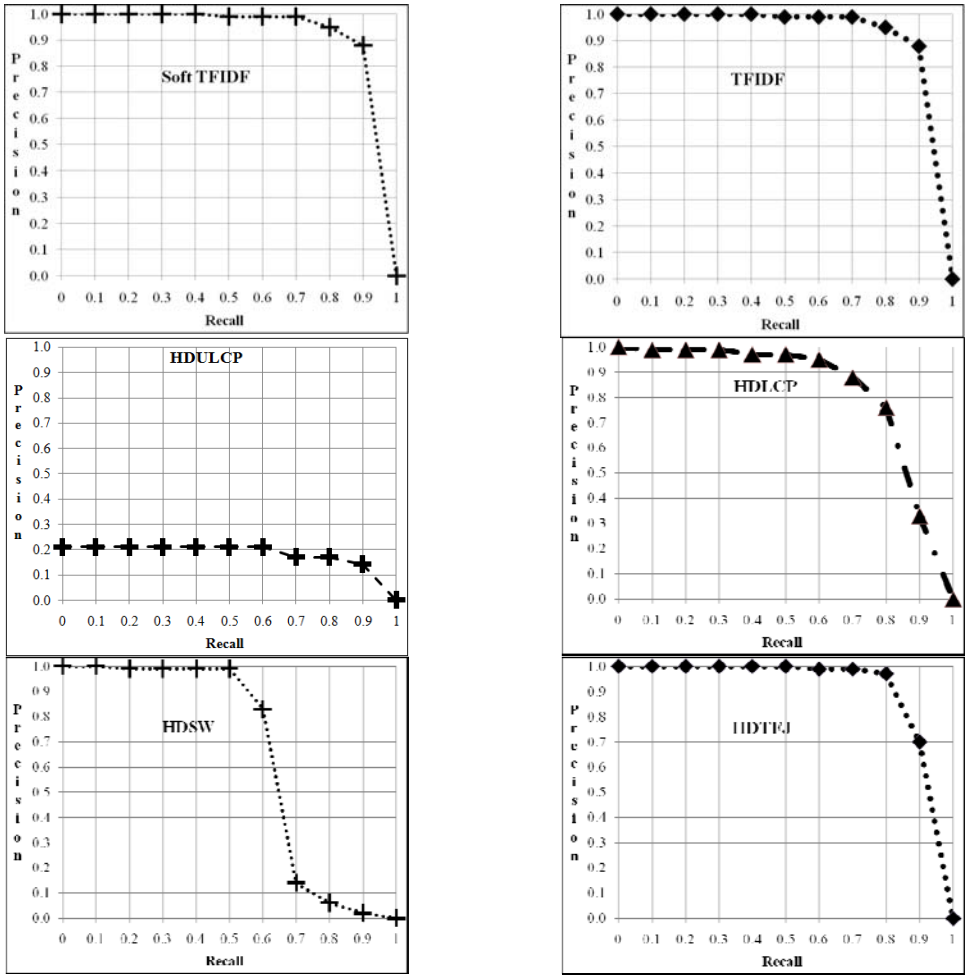


Figure A.2 Precision-recall curves for the duplicate detection experiments on the Animals dataset.

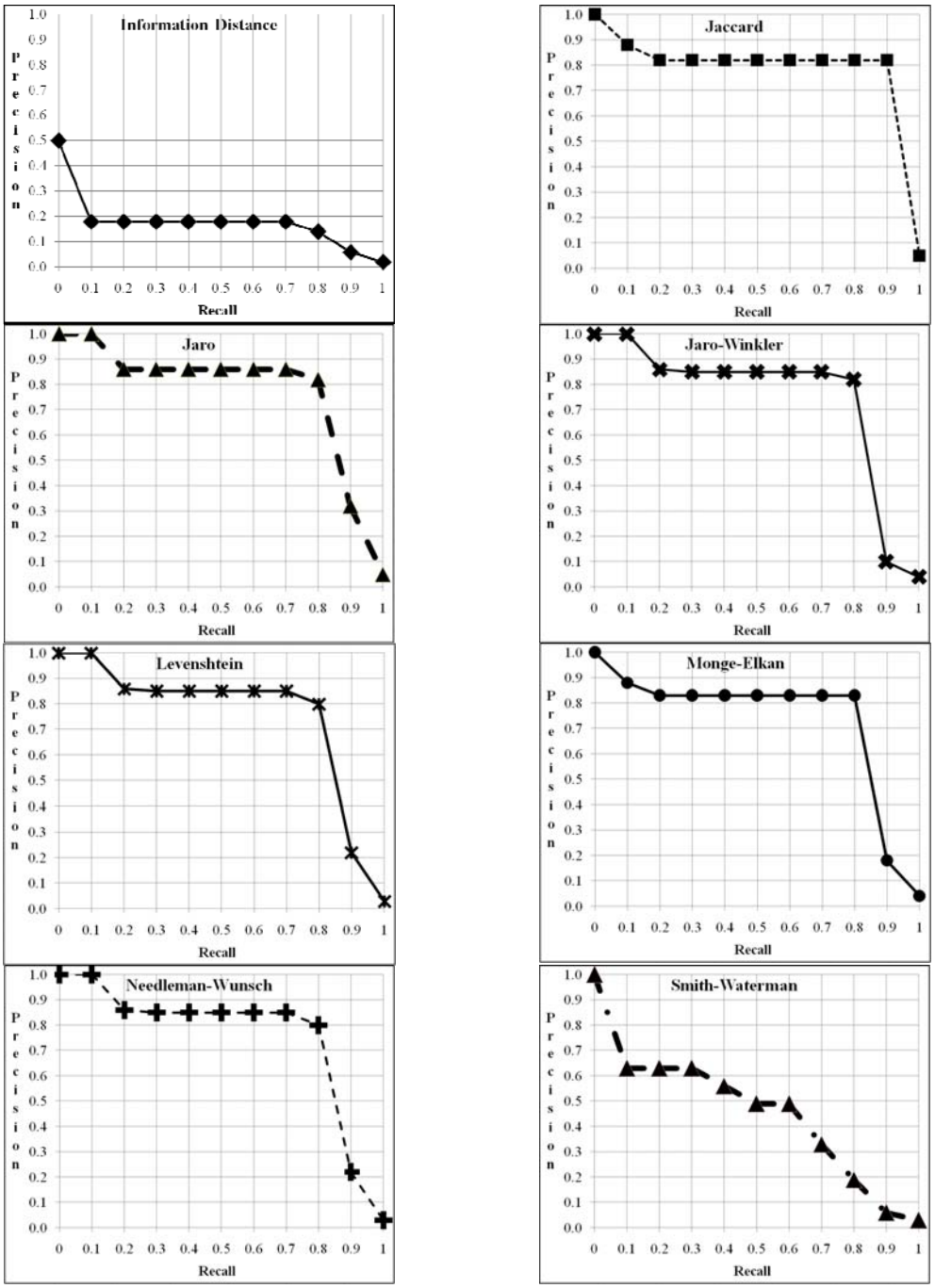


Figure A.3 Precision-recall curves for the duplicate detection experiments on the Birds dataset.

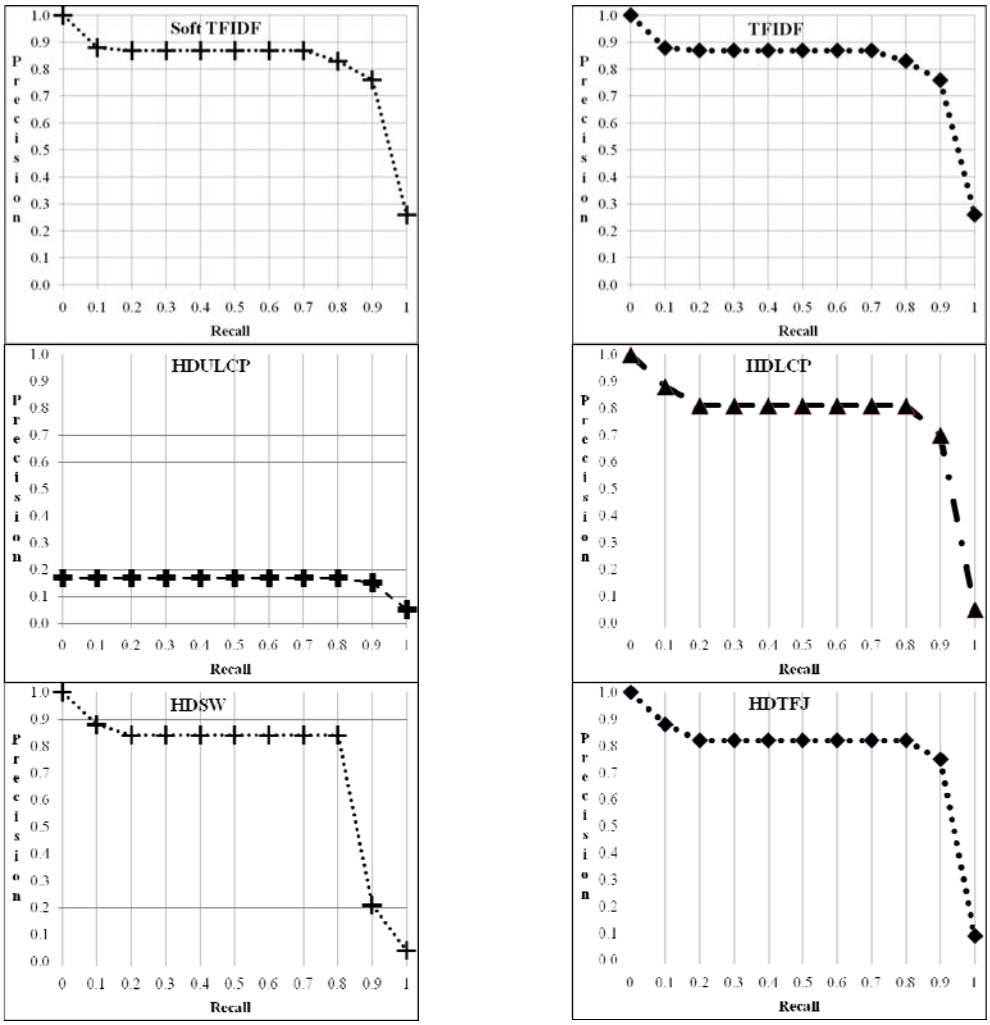


Figure A.4 Precision-recall curves for the duplicate detection experiments on the Birds dataset.

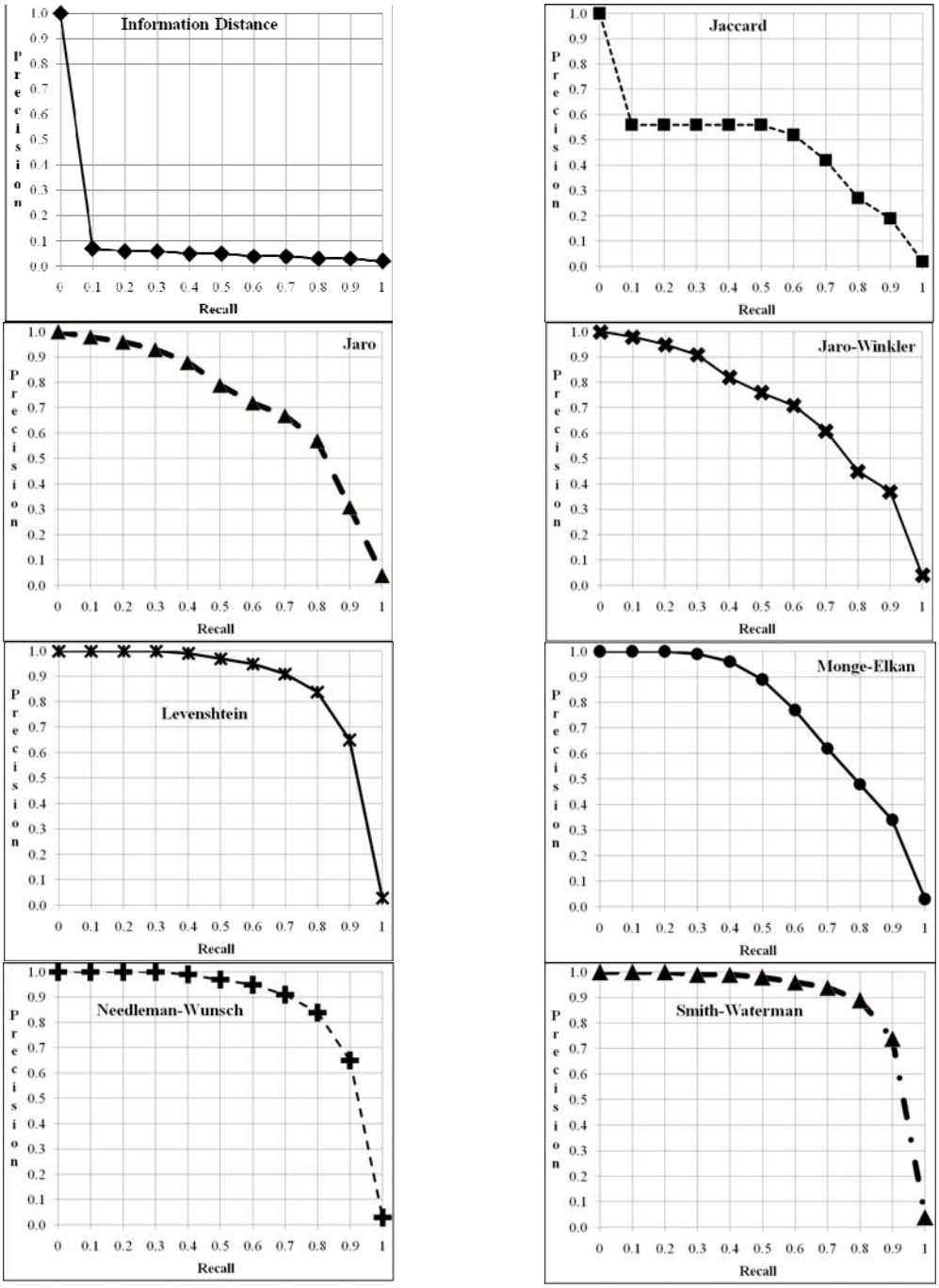


Figure A.5 Precision-recall curves for the duplicate detection experiments on the Census dataset.

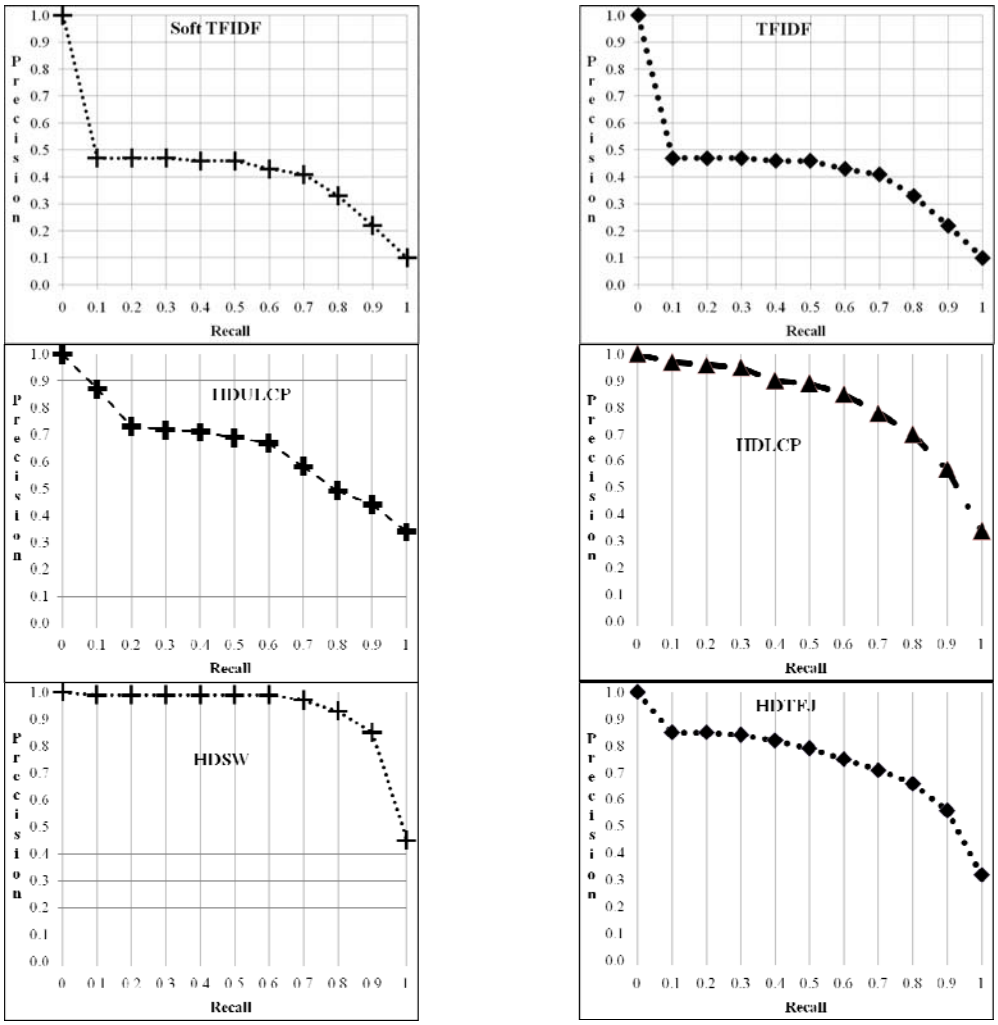


Figure A.6 Precision-recall curves for the duplicate detection experiments on the Census dataset.

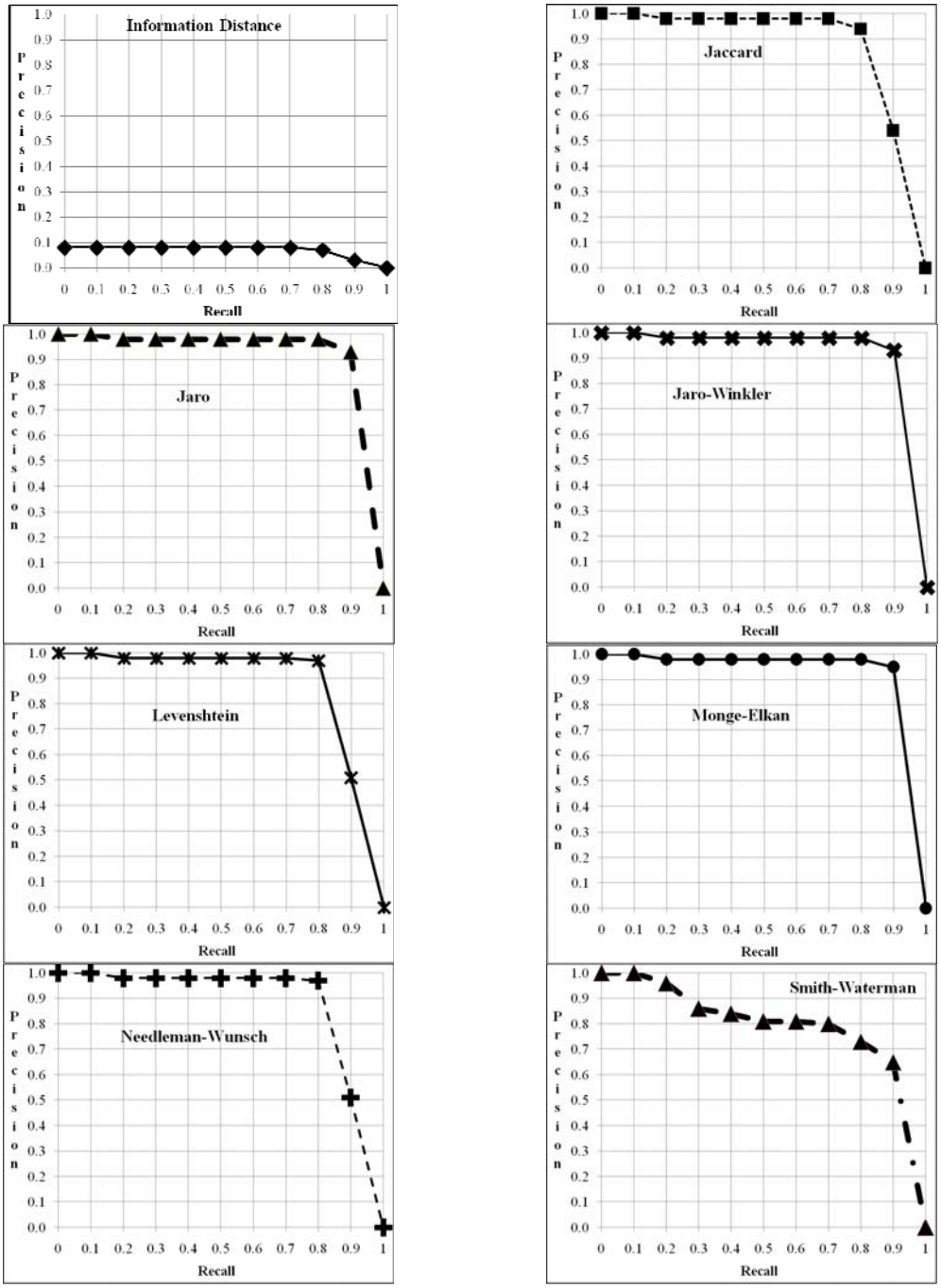


Figure A.7 Precision-recall curves for the duplicate detection experiments on the Parks dataset.

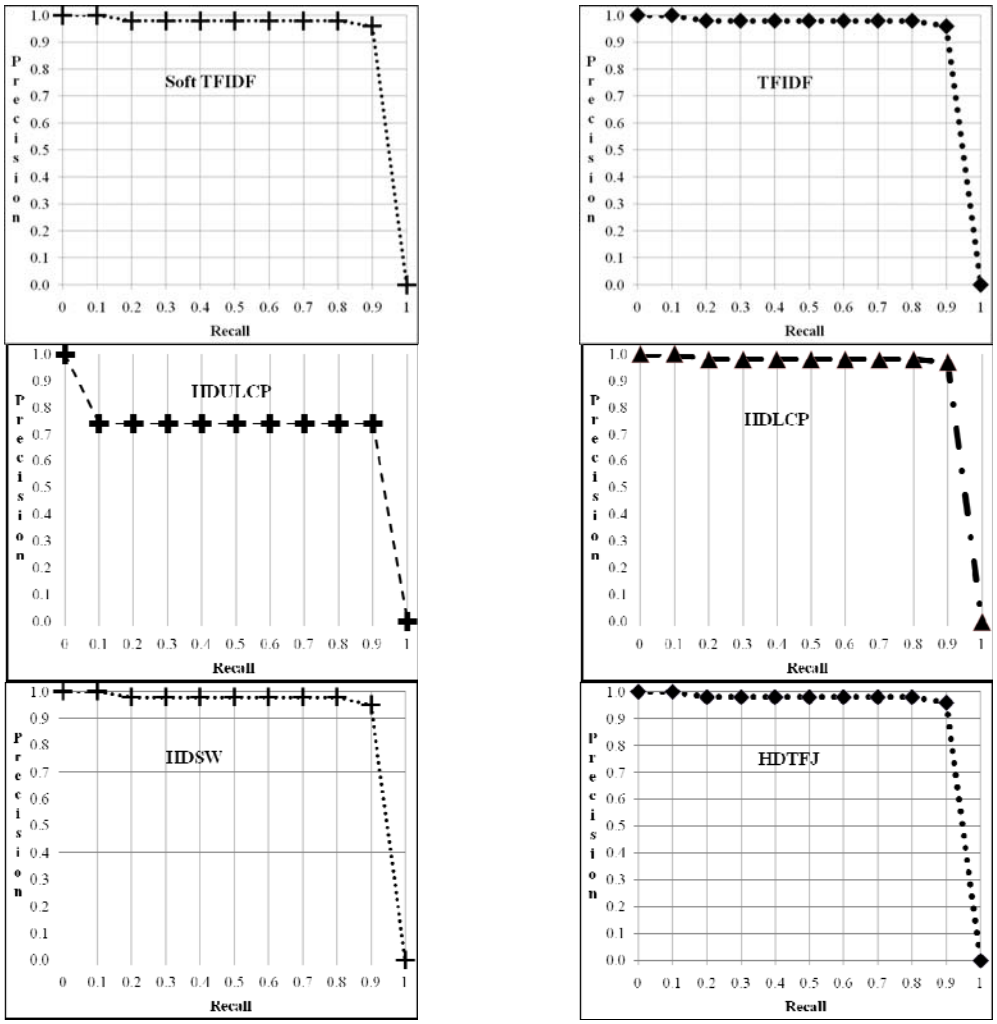


Figure A.8 Precision-recall curves for the duplicate detection experiments on the Parks dataset.

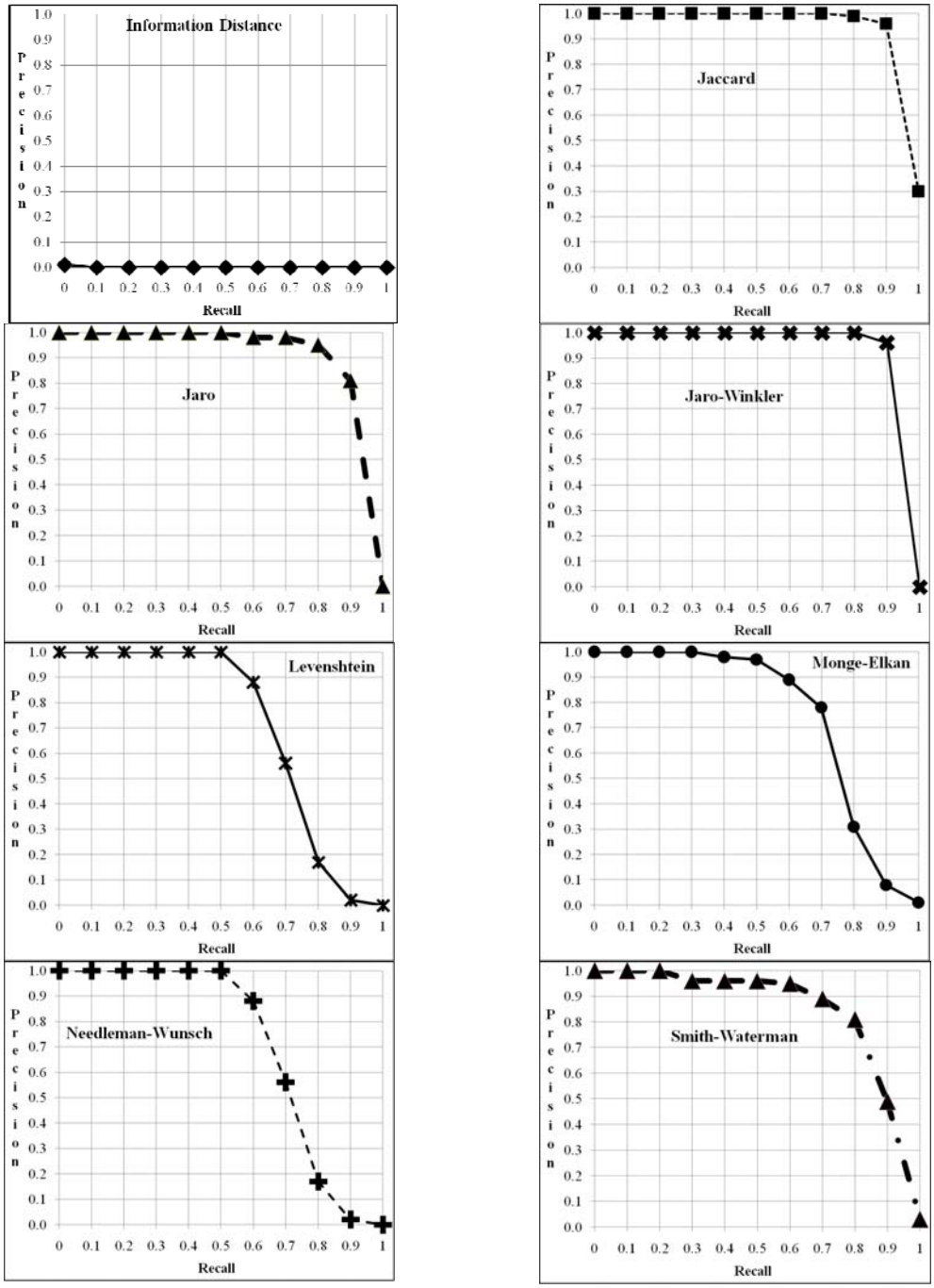


Figure A.9 Precision-recall curves for the duplicate detection experiments on the Restaurants dataset.

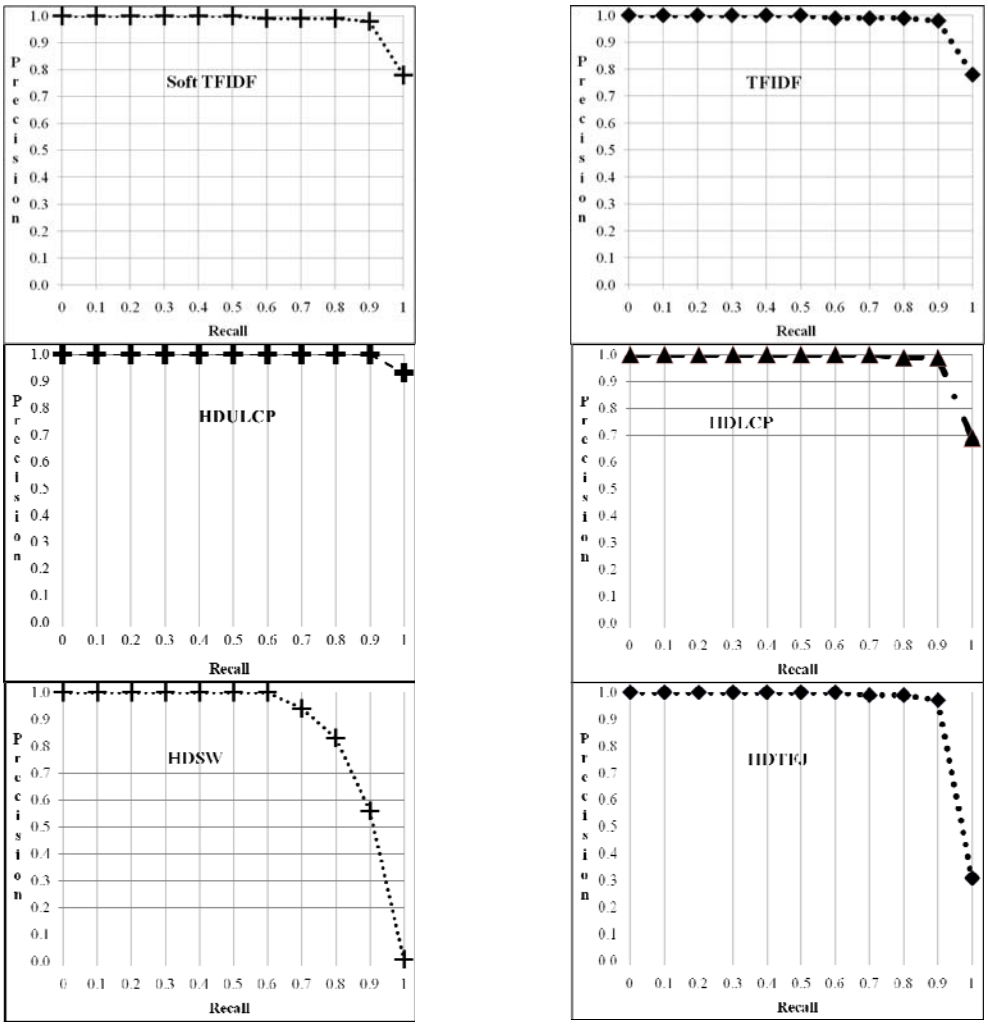


Figure A.10 Precision-recall curves for the duplicate detection experiments on the Restaurants dataset.

APPENDIX B

PRECISION-RECALL CHARTS FOR THE CLUSTERING EXPERIMENTS ON LIFE AND SOCIAL SCIENCES DATASETS

This appendix provides precision-recall charts for the evaluation presented in the Section 5.3.4.

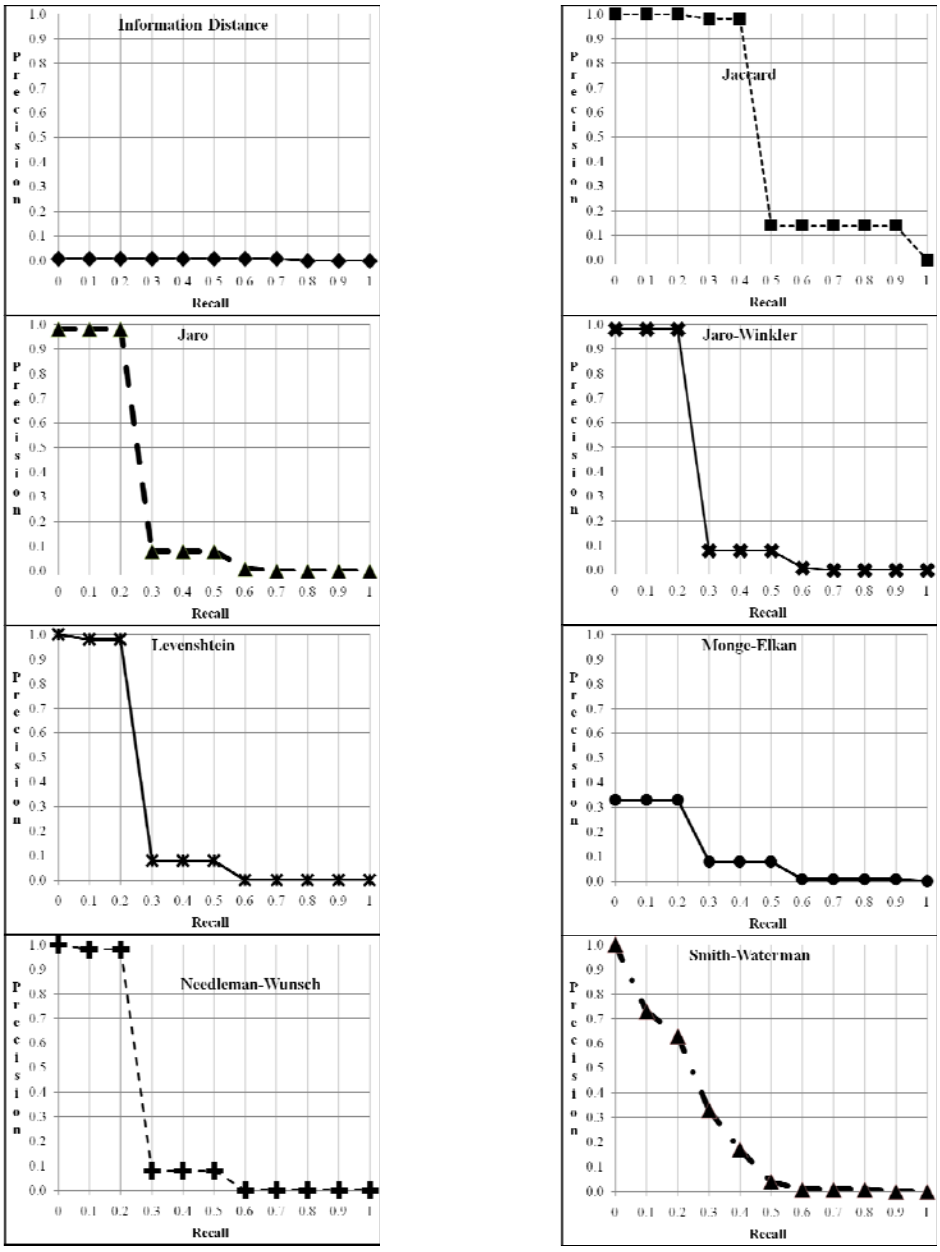


Figure B.1 Precision-recall curves for the clustering experiments on the Animals dataset.

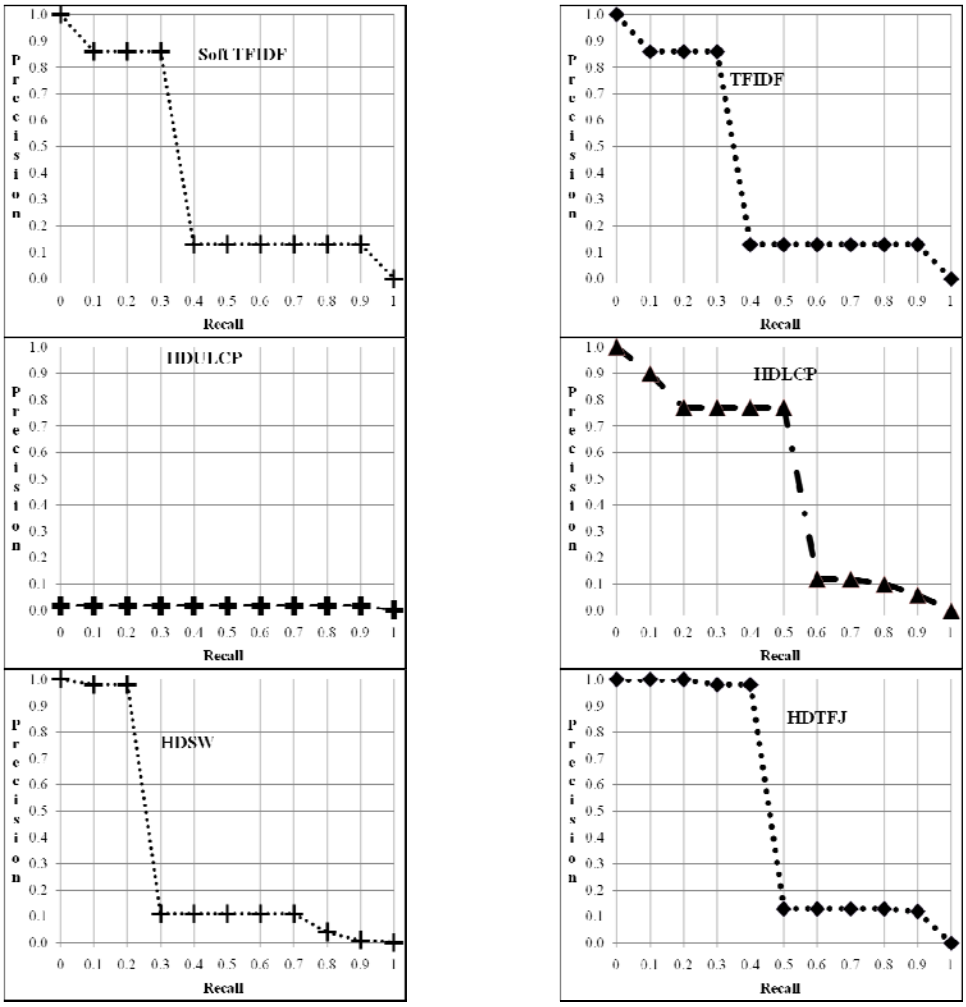


Figure B.2 Precision-recall curves for the clustering experiments on the Animals dataset.

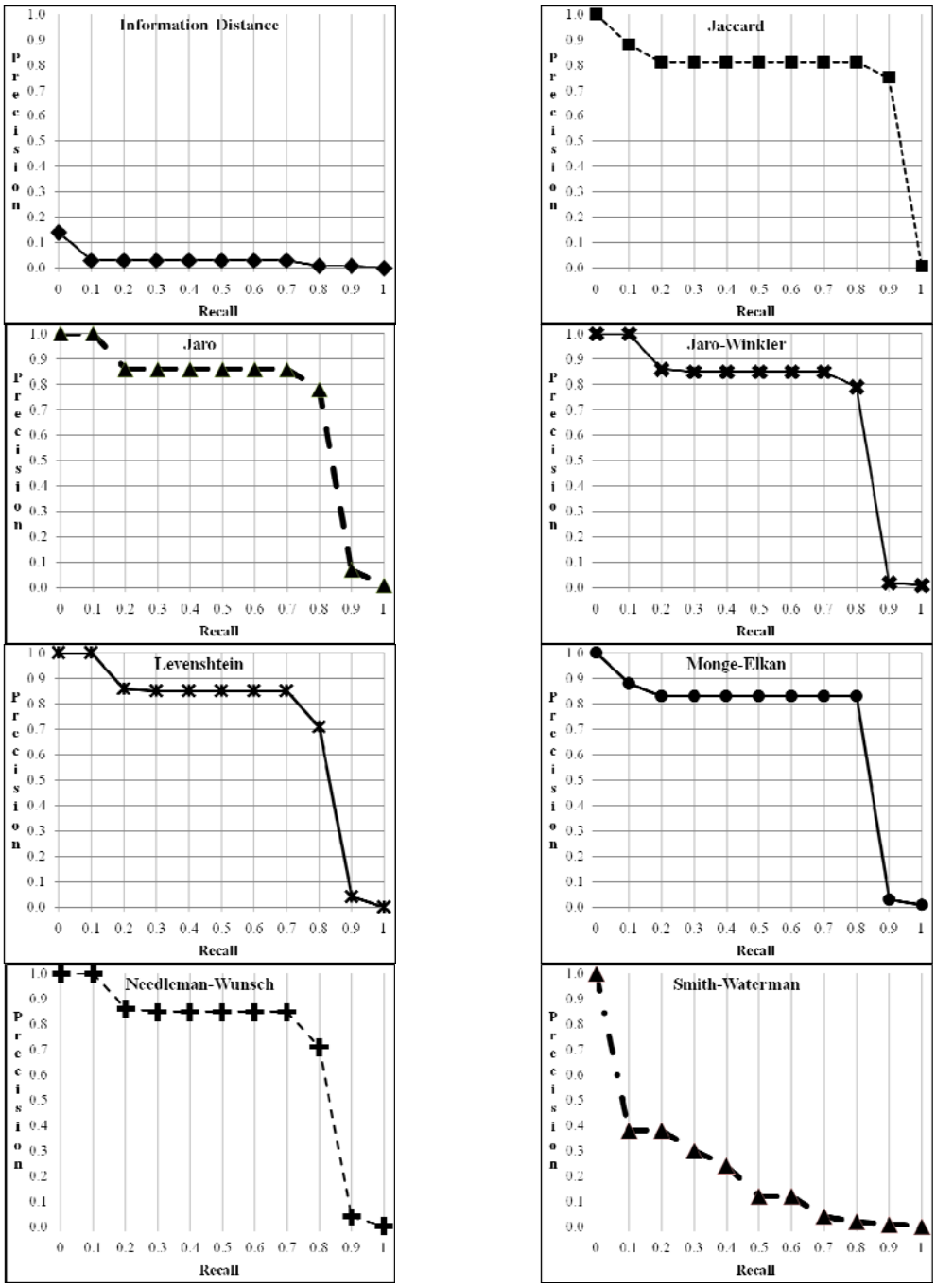


Figure B.3 Precision-recall curves for the clustering experiments on the Birds dataset.

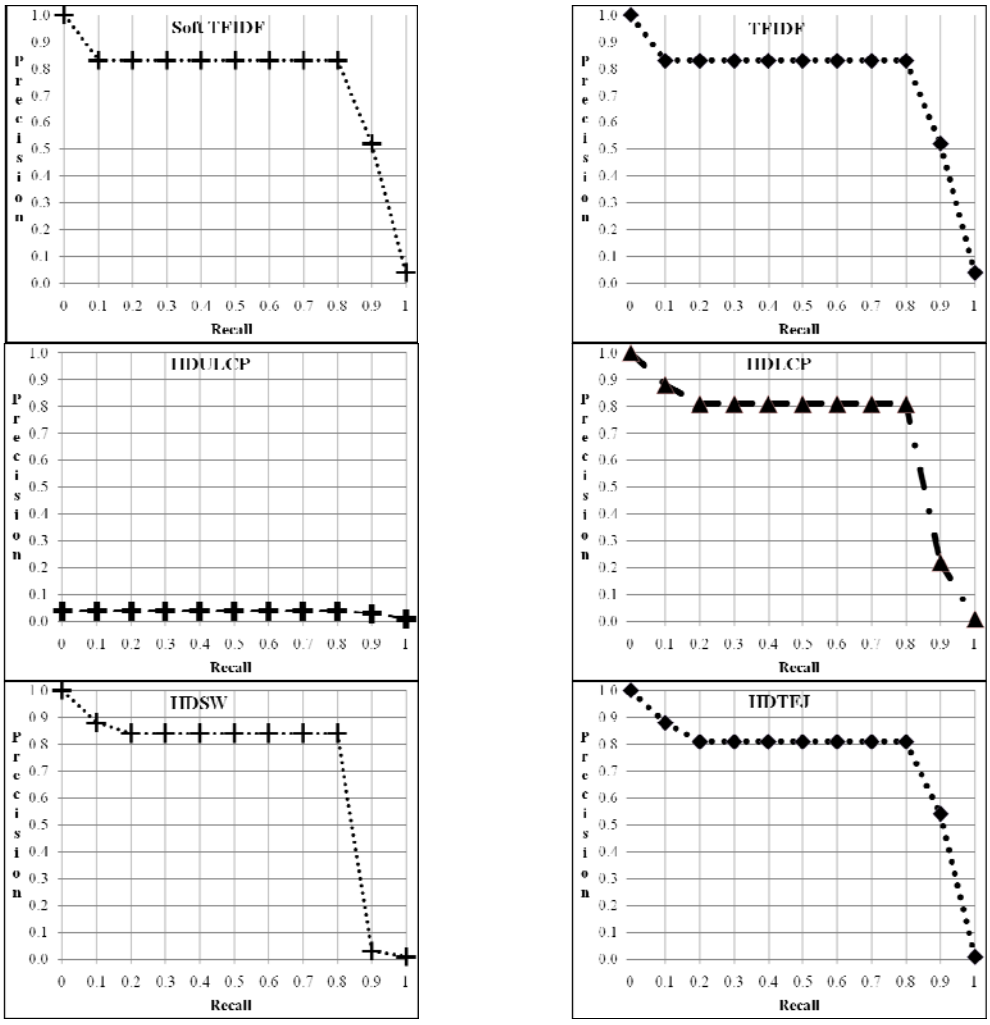


Figure B.4 Precision-recall curves for the clustering experiments on the Birds dataset.

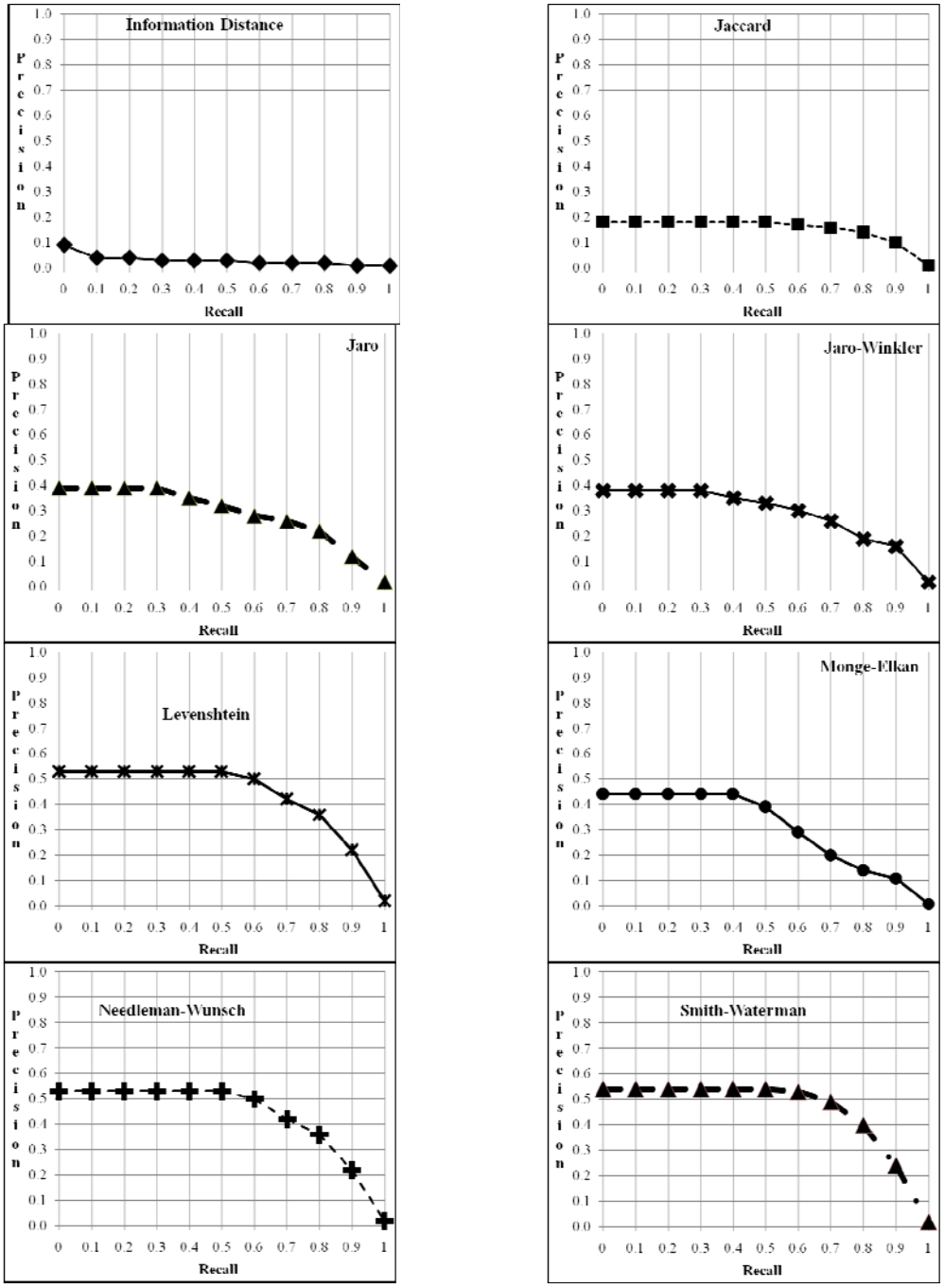


Figure B.5 Precision-recall curves for the clustering experiments on the Census dataset.

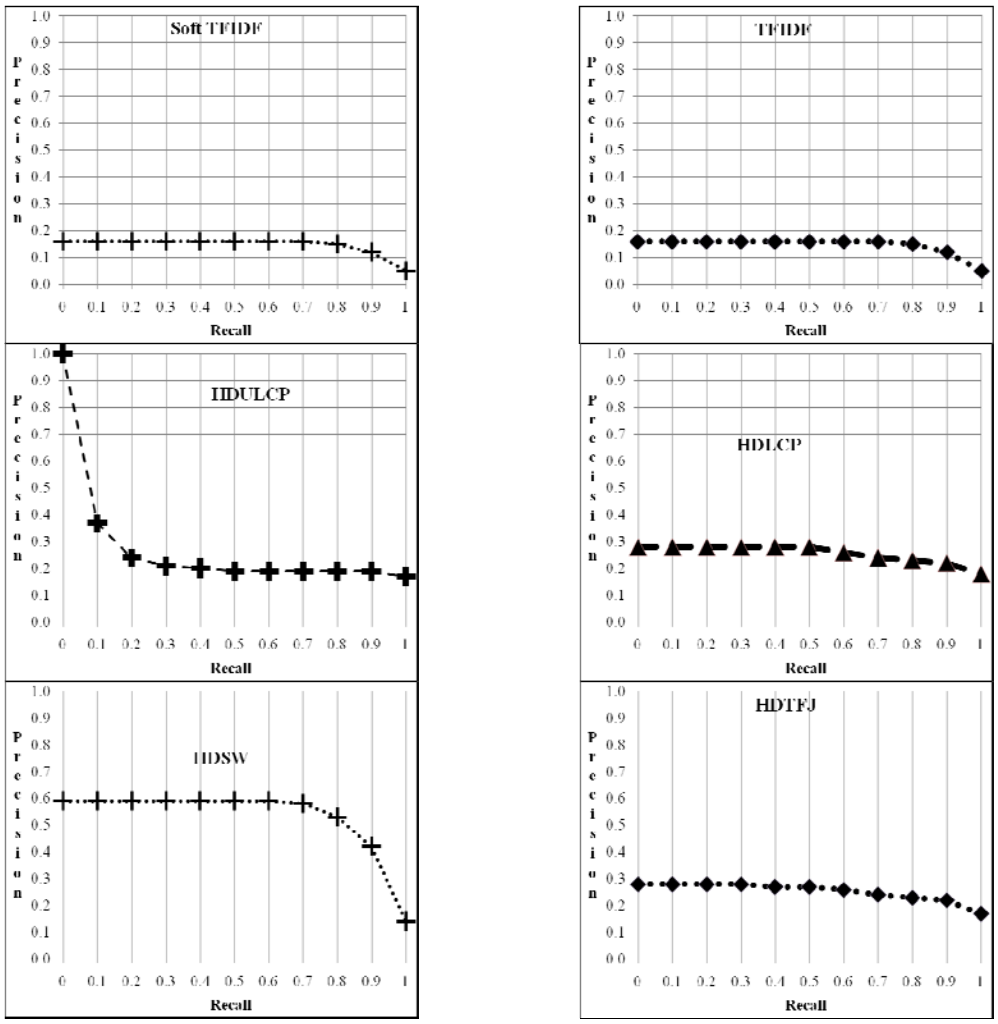


Figure B.6 Precision-recall curves for the clustering experiments on the Census dataset.

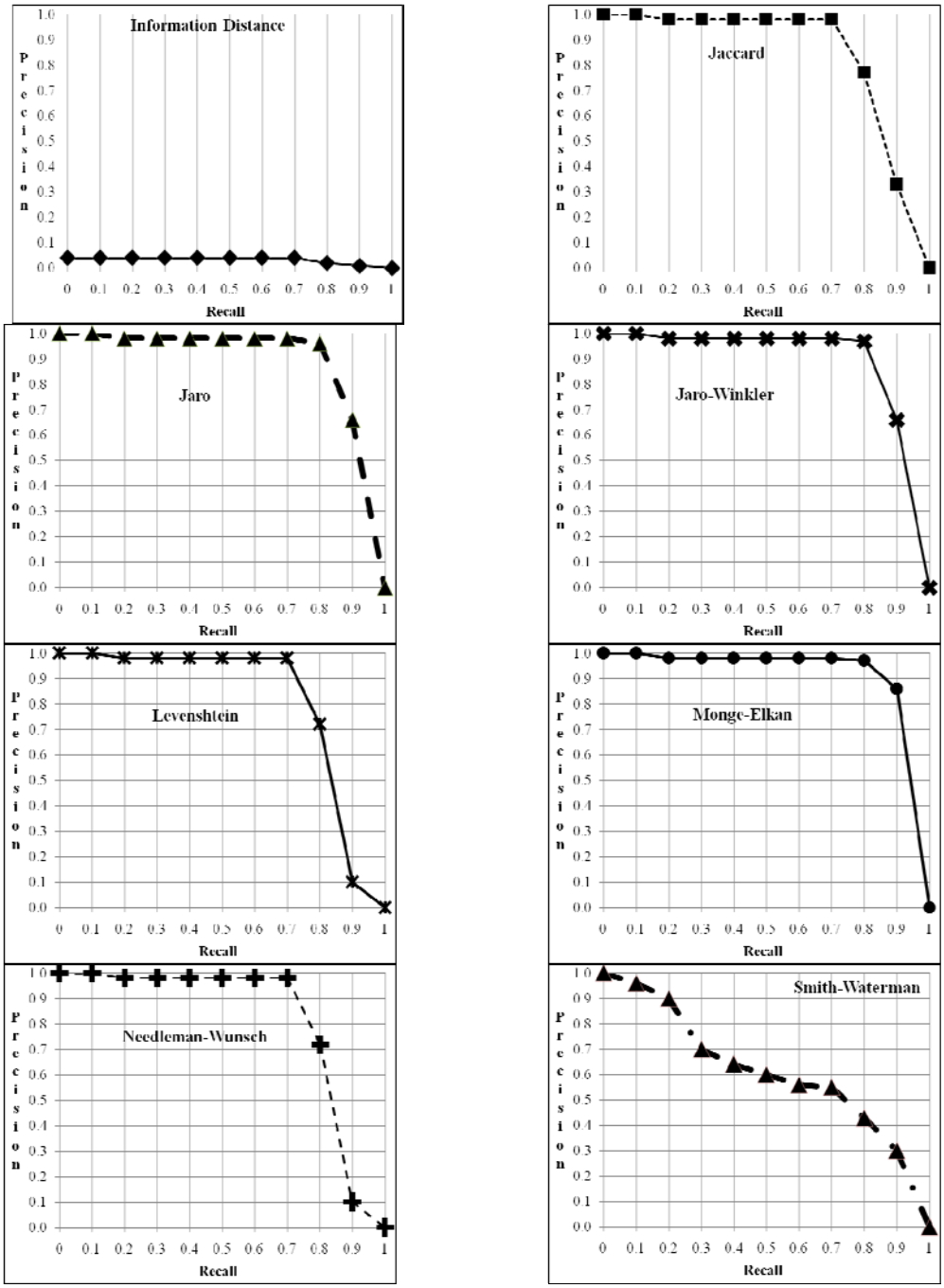


Figure B.7 Precision-recall curves for the clustering experiments on the Parks dataset.

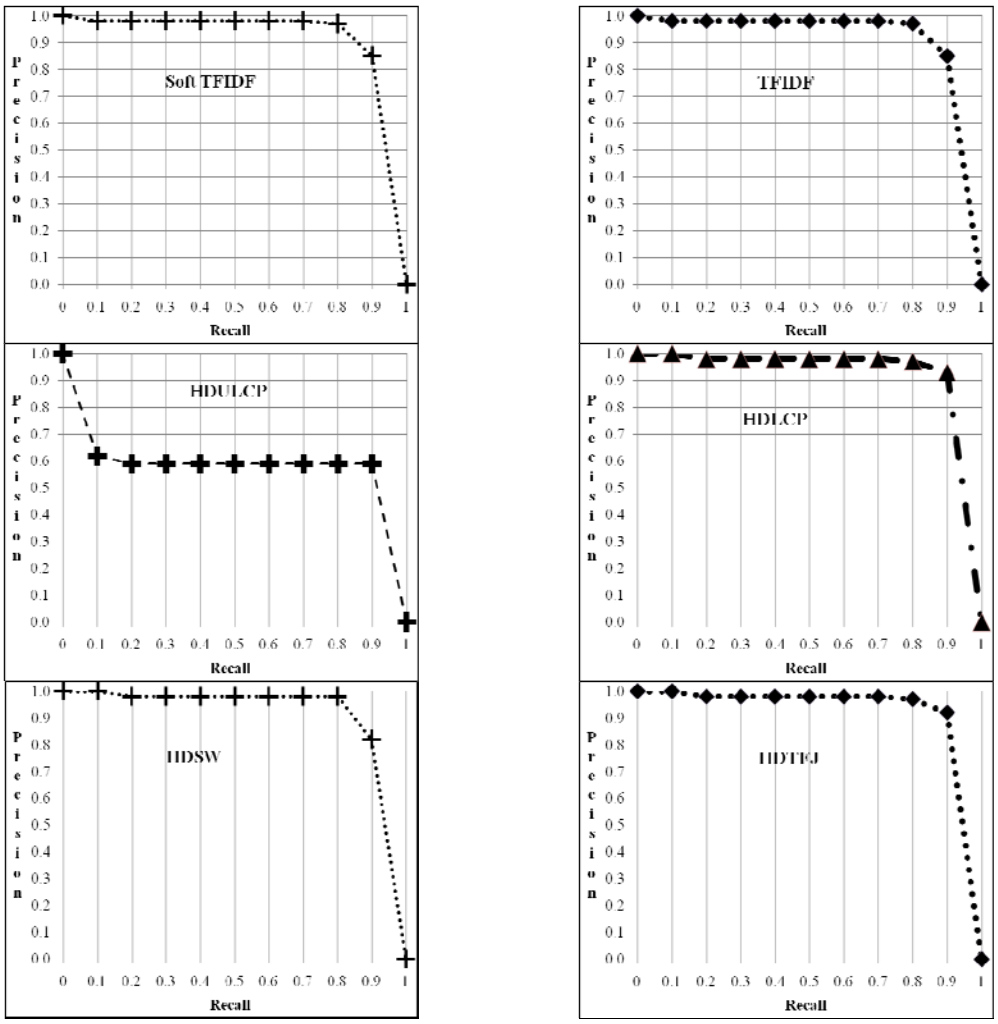


Figure B.8 Precision-recall curves for the clustering experiments on the Parks dataset.

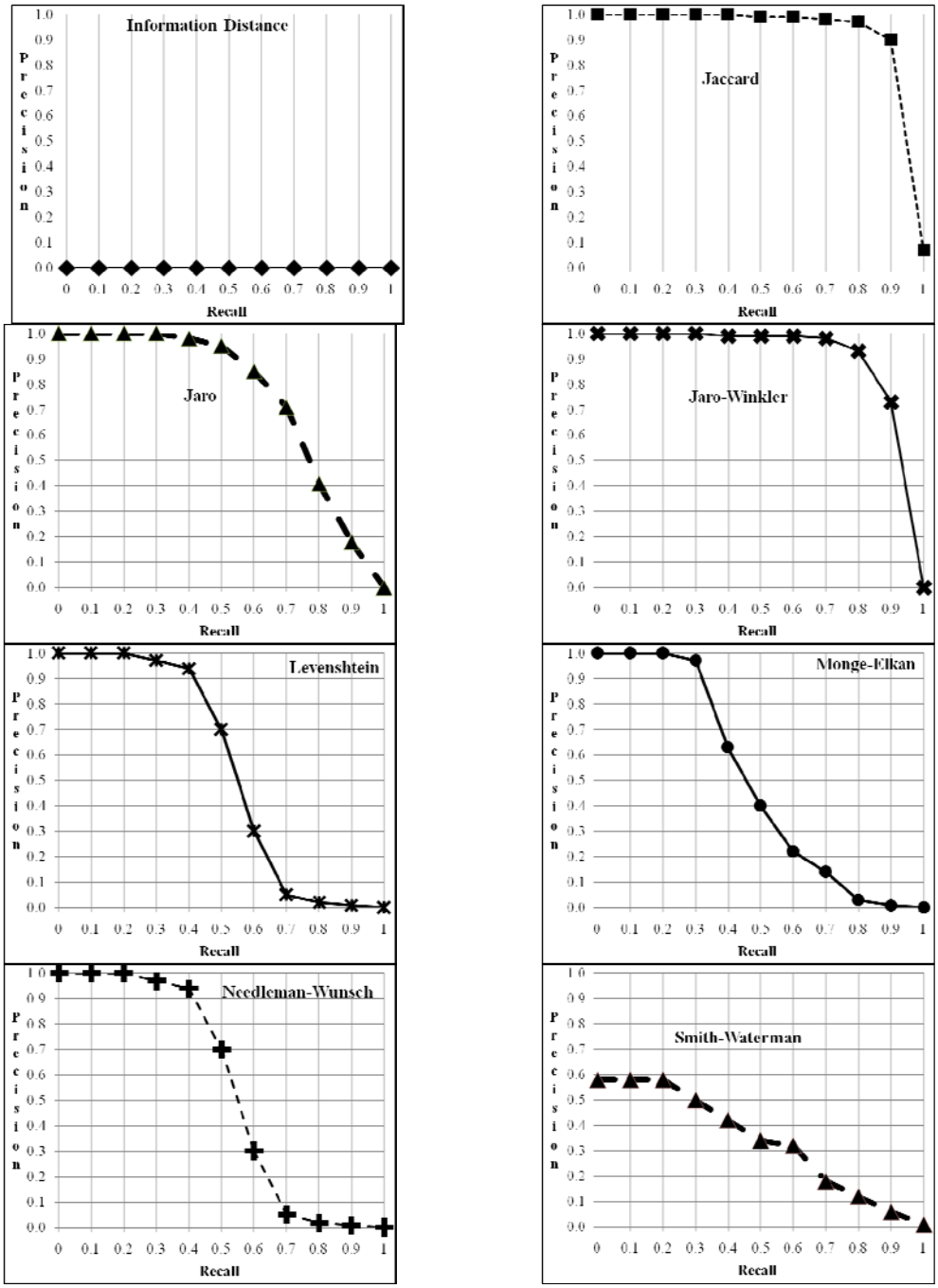


Figure B.9 Precision-recall curves for the clustering experiments on the Restaurants dataset.

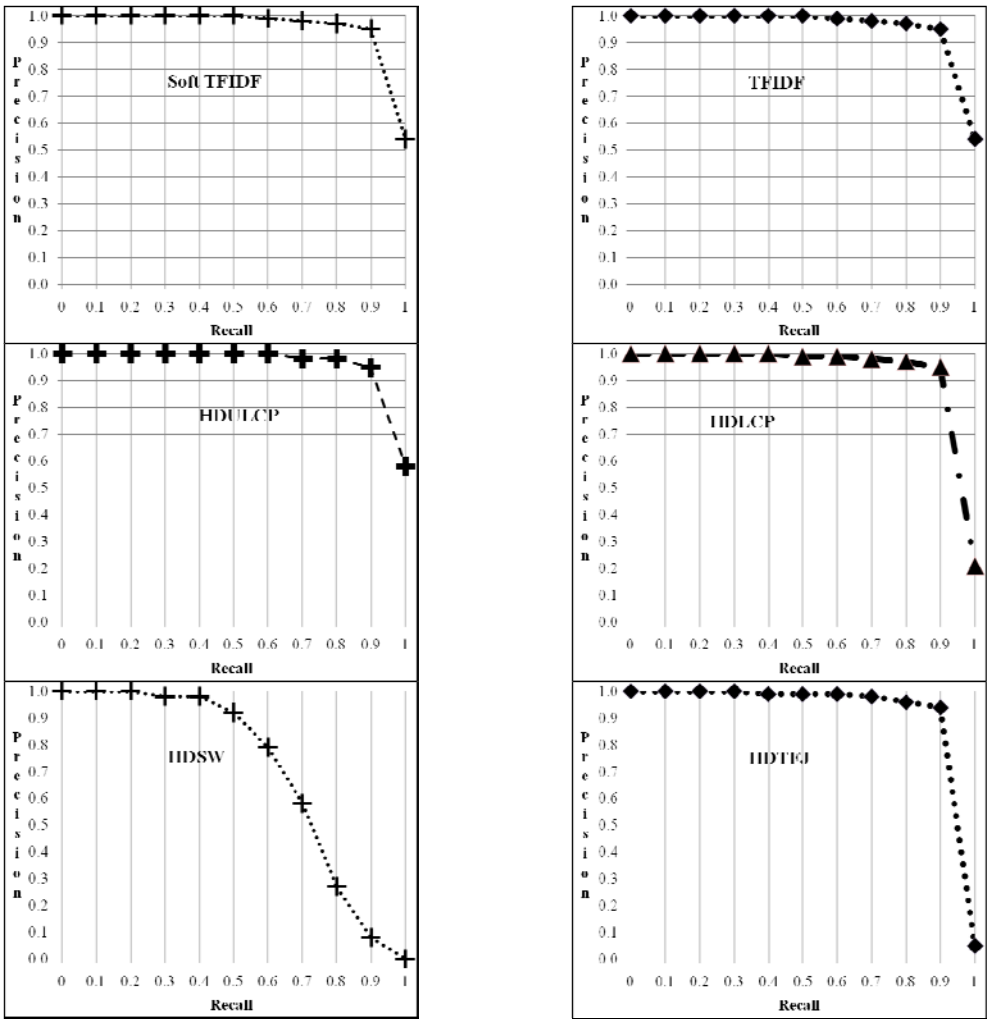


Figure B.10 Precision-recall curves for the clustering experiments on the Restaurants dataset.

APPENDIX C

PRECISION-RECALL CHARTS FOR DUPLICATE DETECTION EXPERIMENTS ON BIOINFORMATICS DATASETS

This appendix provides precision-recall charts for the evaluation presented in the Section 5.4.4.

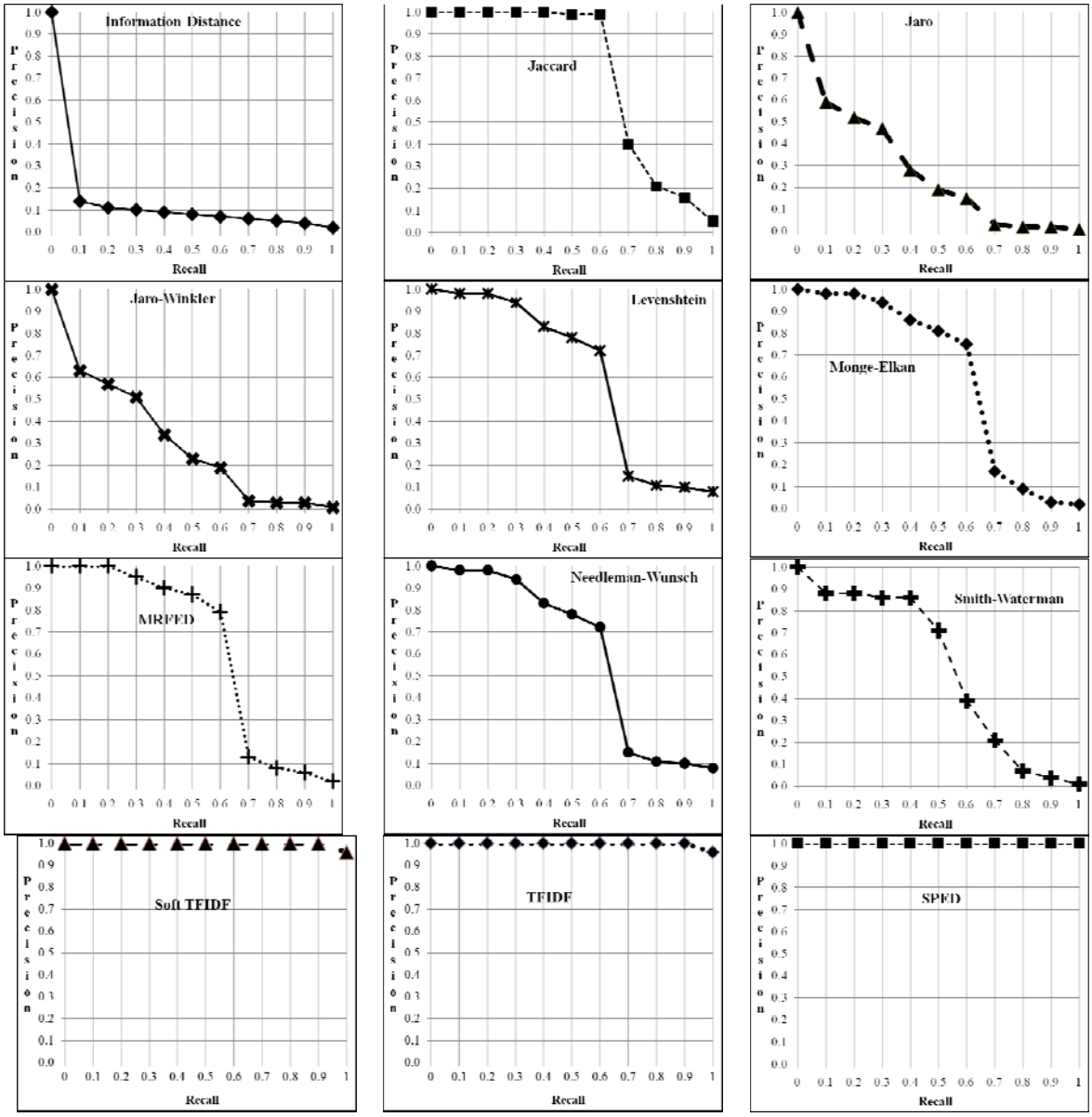


Figure C.1 Precision-recall curves for duplicate detection experiments on the Paramecium Tetraurelia dataset.

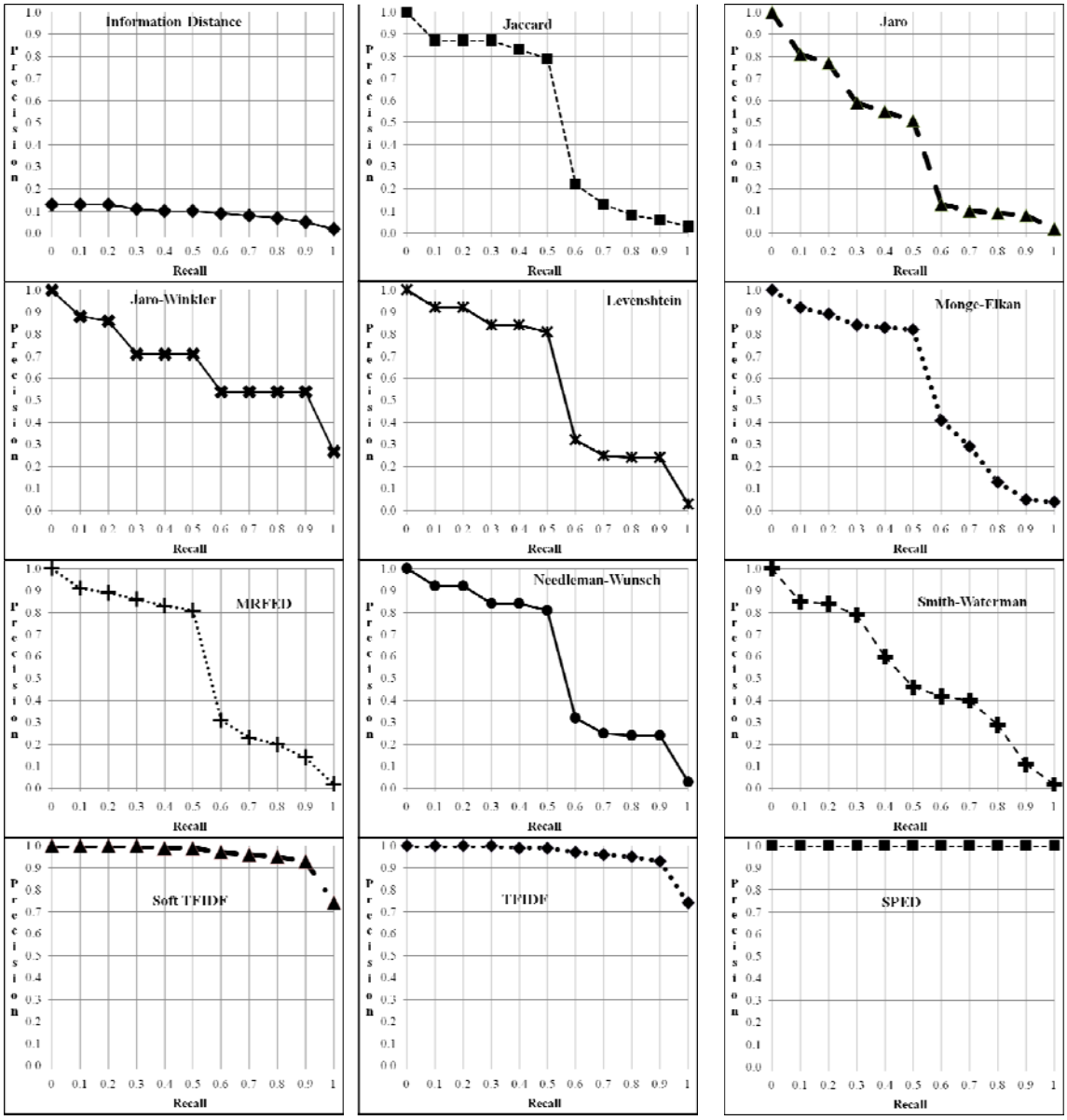


Figure C.2 Precision-recall curves for duplicate detection experiments on the Bacteriophage T4 dataset.

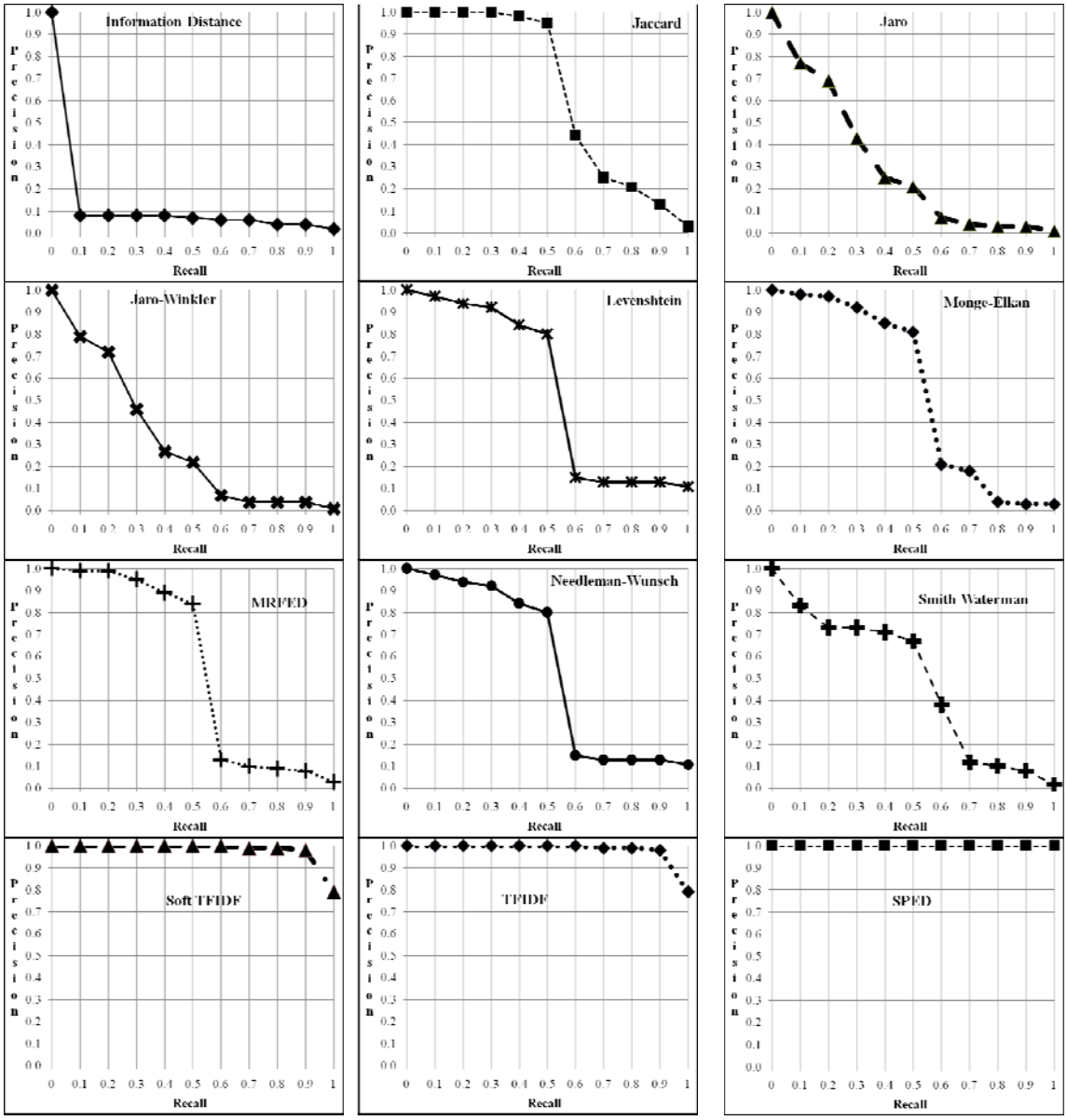


Figure C.3 Precision-recall curves for duplicate detection experiments on the Carsonella Ruddii dataset.

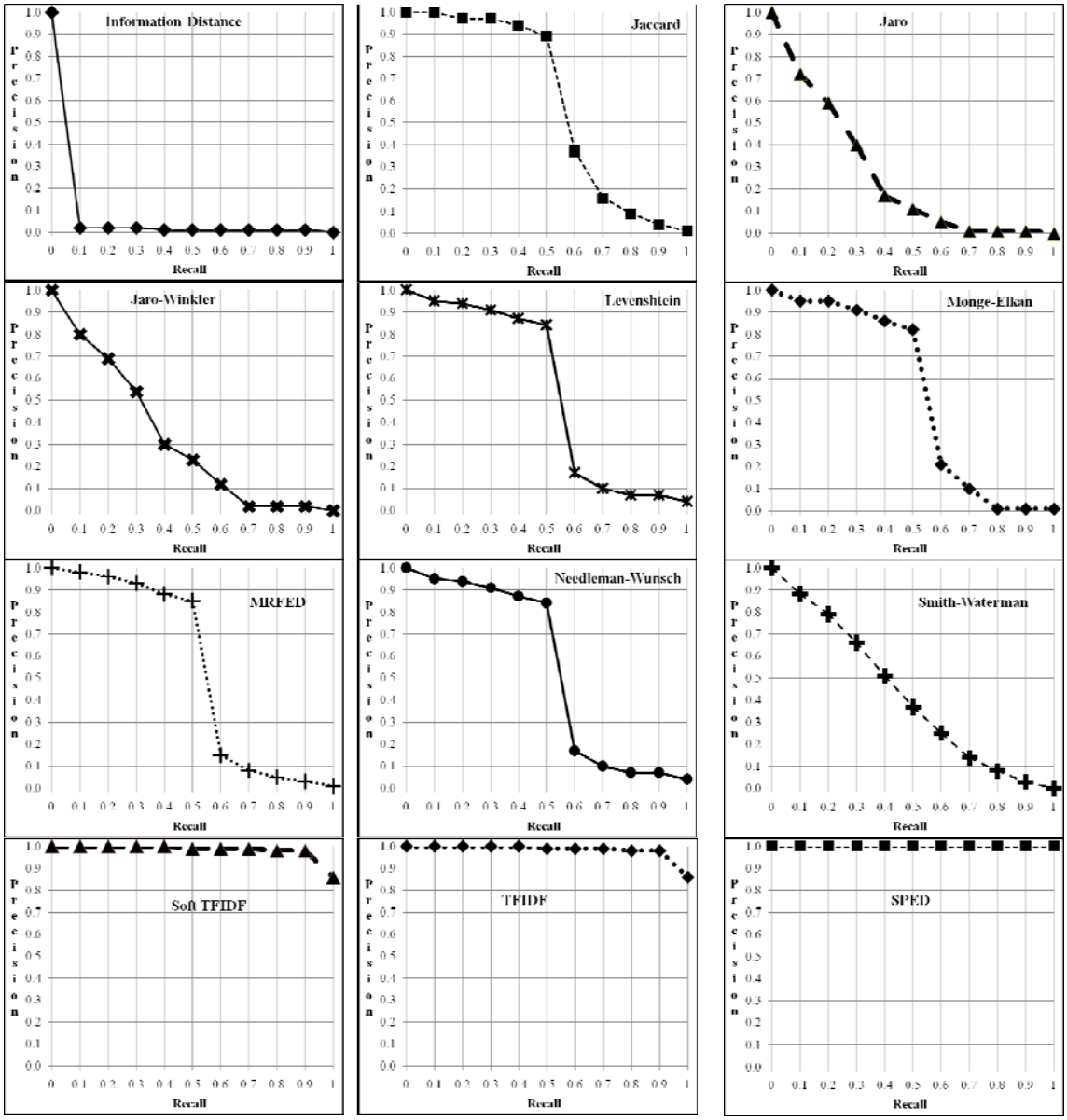


Figure C.4 Precision-recall curves for duplicate detection experiments on the Hyperthermus Butylicus dataset.

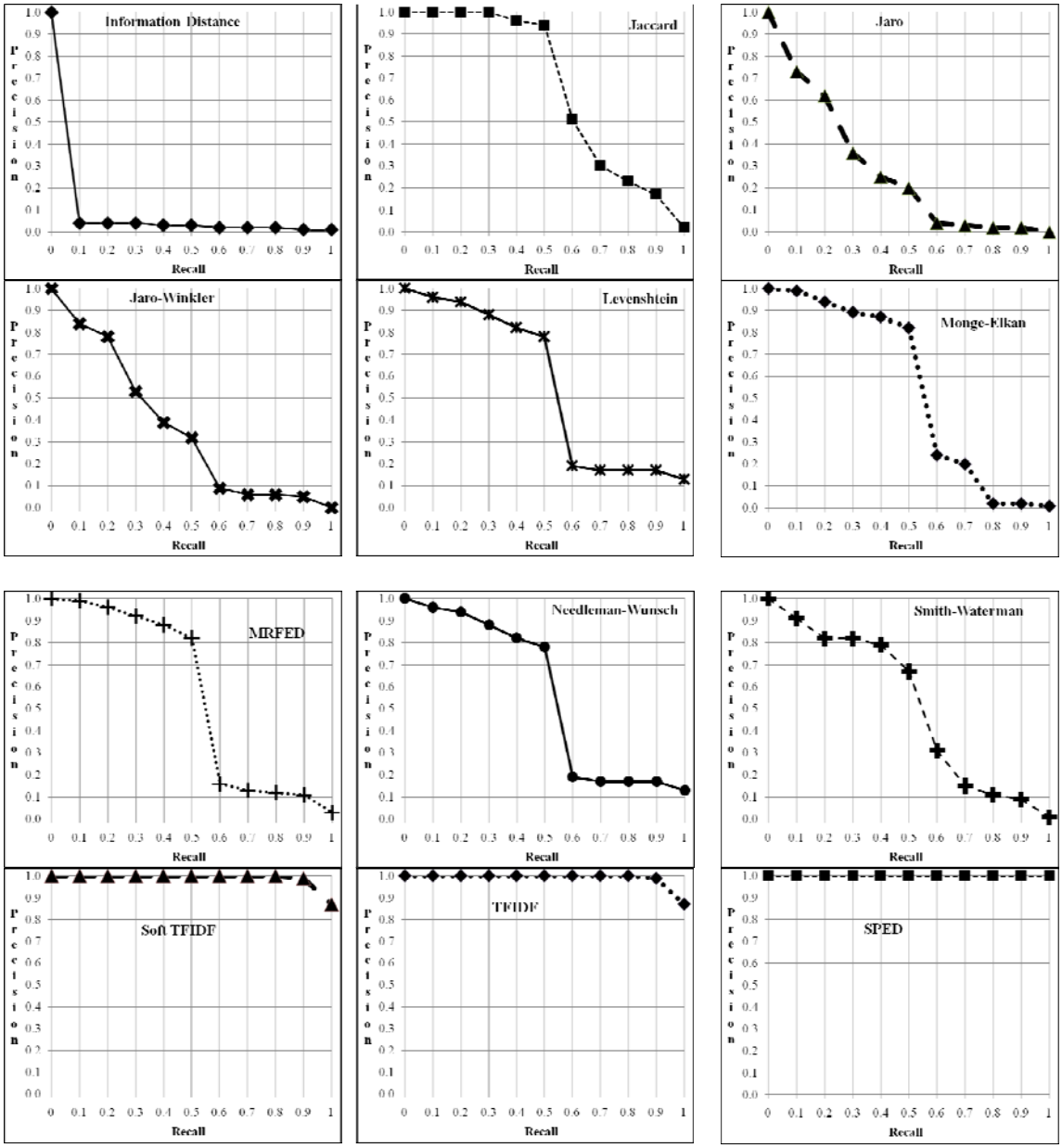


Figure C.5 Precision-recall curves for duplicate detection experiments on the Buchnera Aphidicola Cedri Cinara dataset.

APPENDIX D

PRECISION-RECALL CHARTS FOR CLUSTERING EXPERIMENTS ON BIOINFORMATICS DATASETS

This appendix provides precision-recall charts for the evaluation presented in the Section 5.5.4.

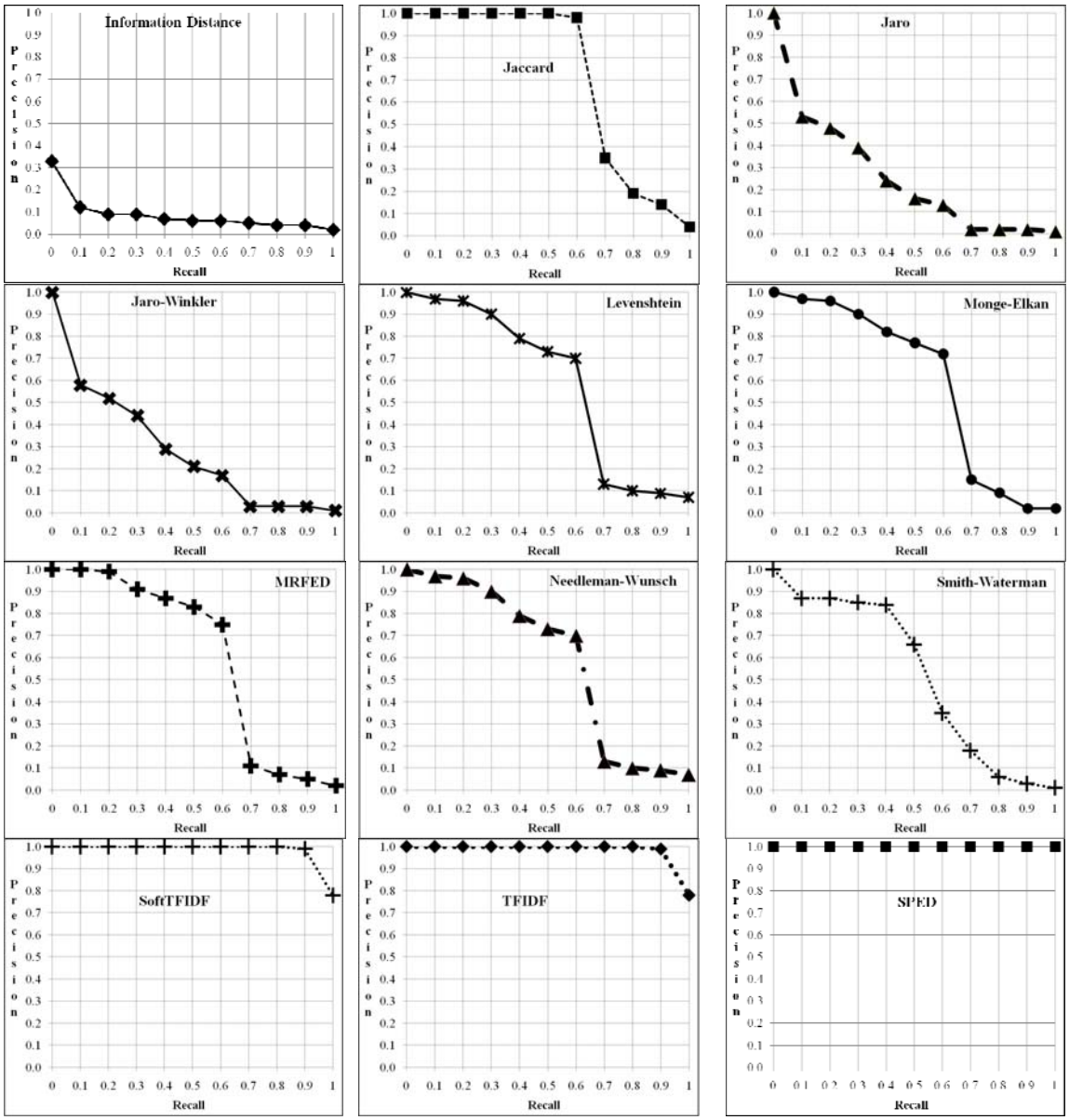


Figure D.1 Precision-recall curves for clustering experiments on the Paramecium Tetraurelia dataset.

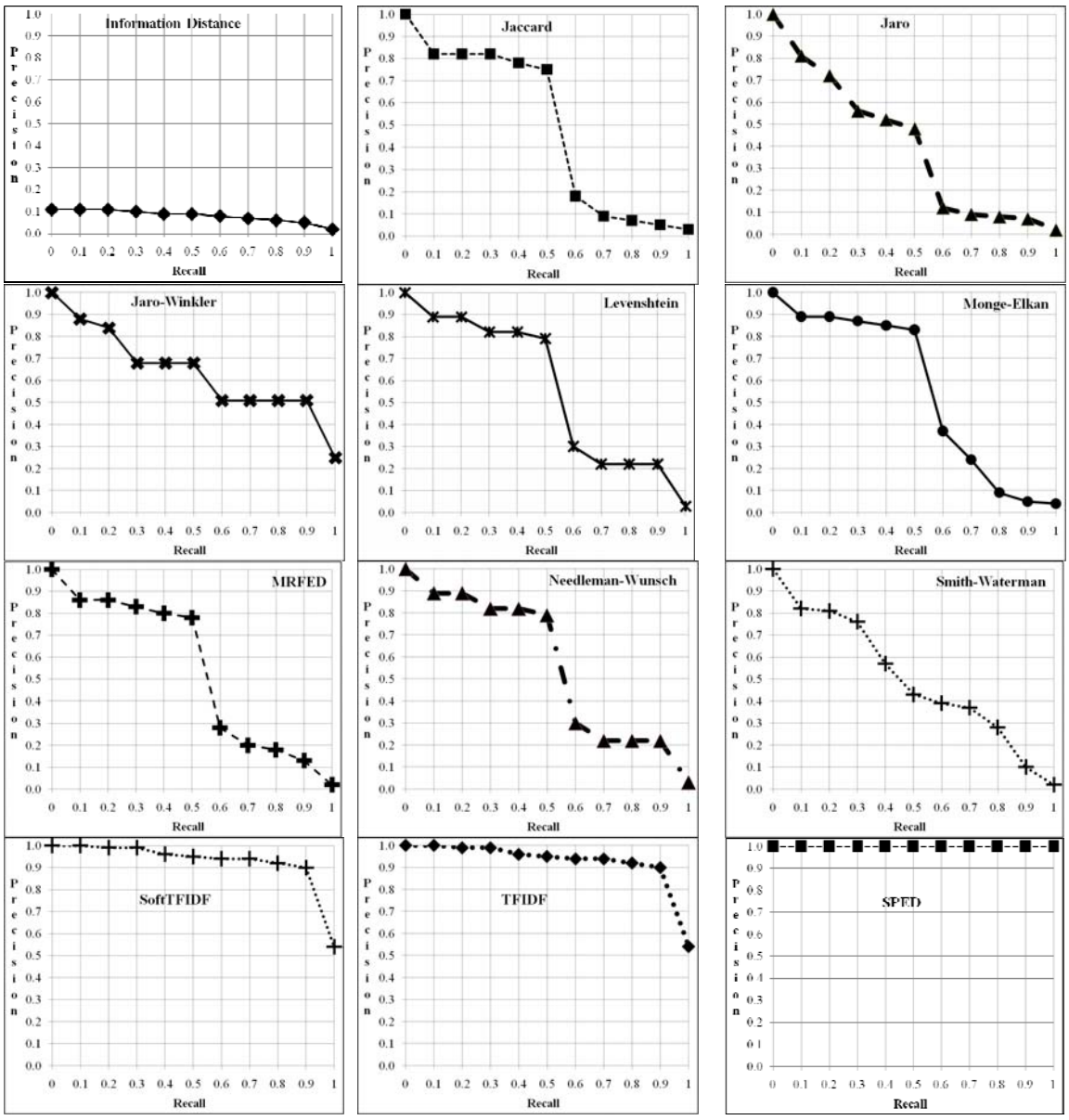


Figure D.2 Precision-recall curves for clustering experiments on the Bacteriophage T4 dataset.

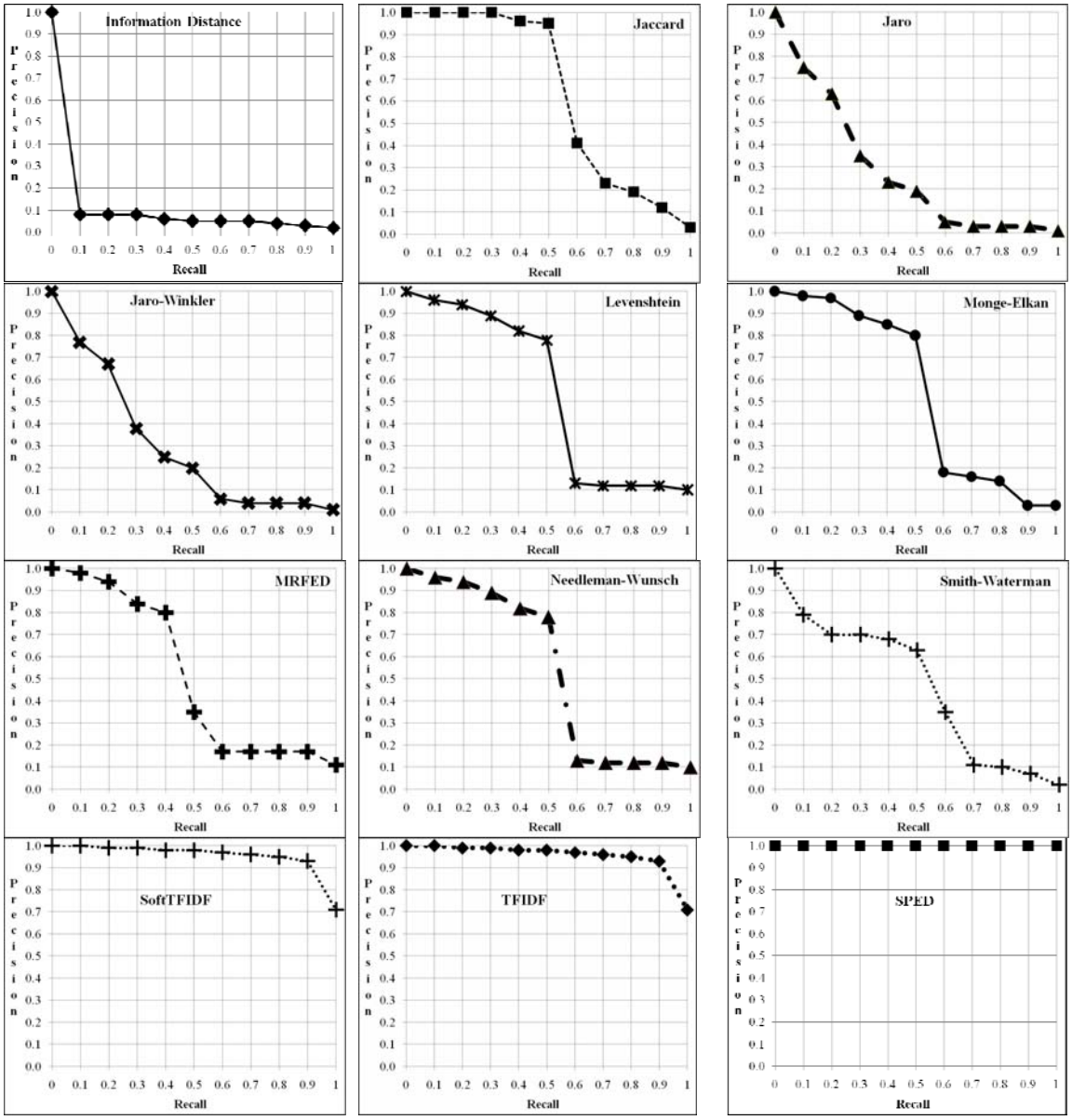


Figure D.3 Precision-recall curves for clustering experiments on the Carsonella Ruddii dataset.

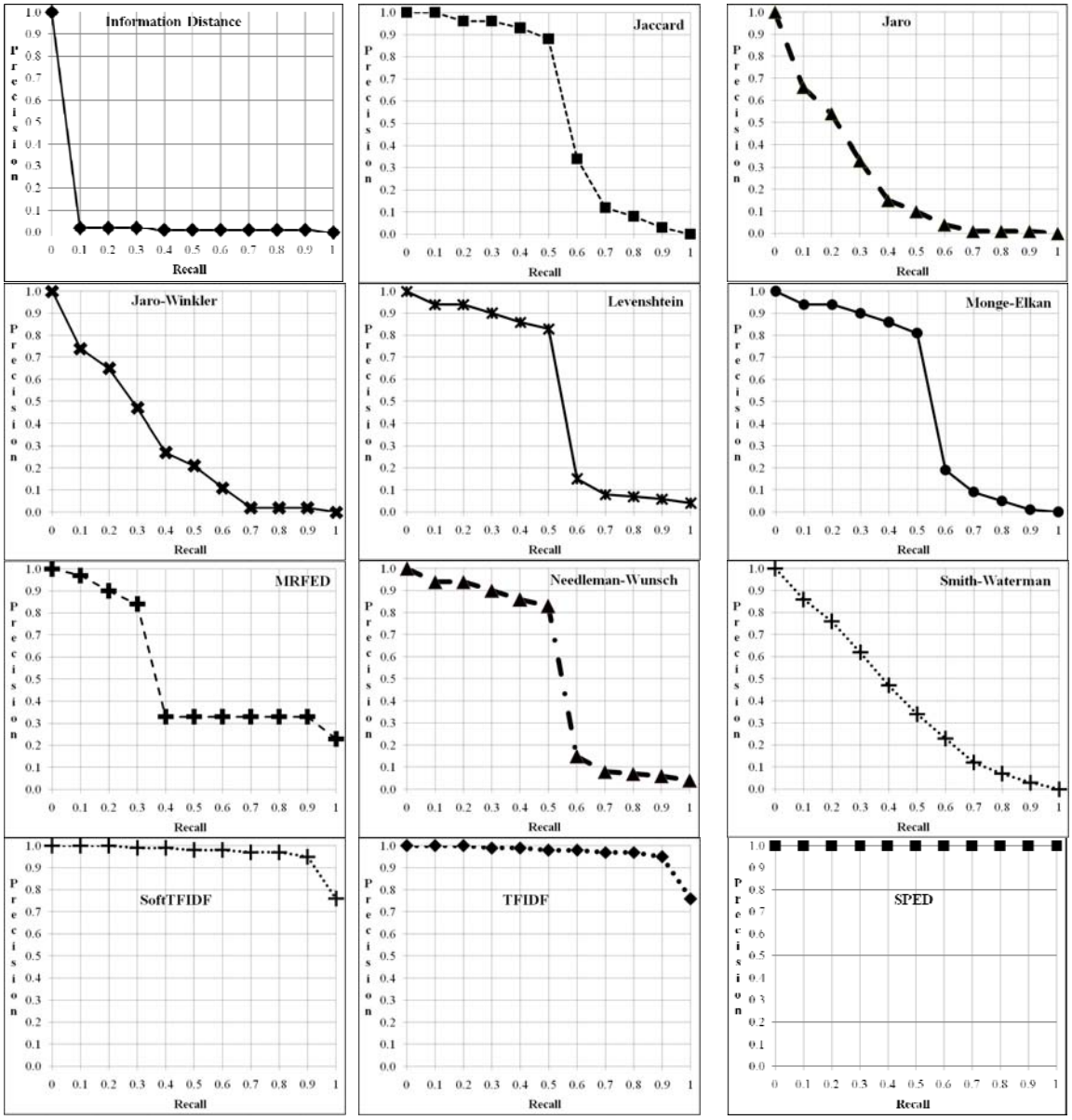


Figure D.4 Precision-recall curves for clustering experiments on the Hyperthermus Butylicus dataset.

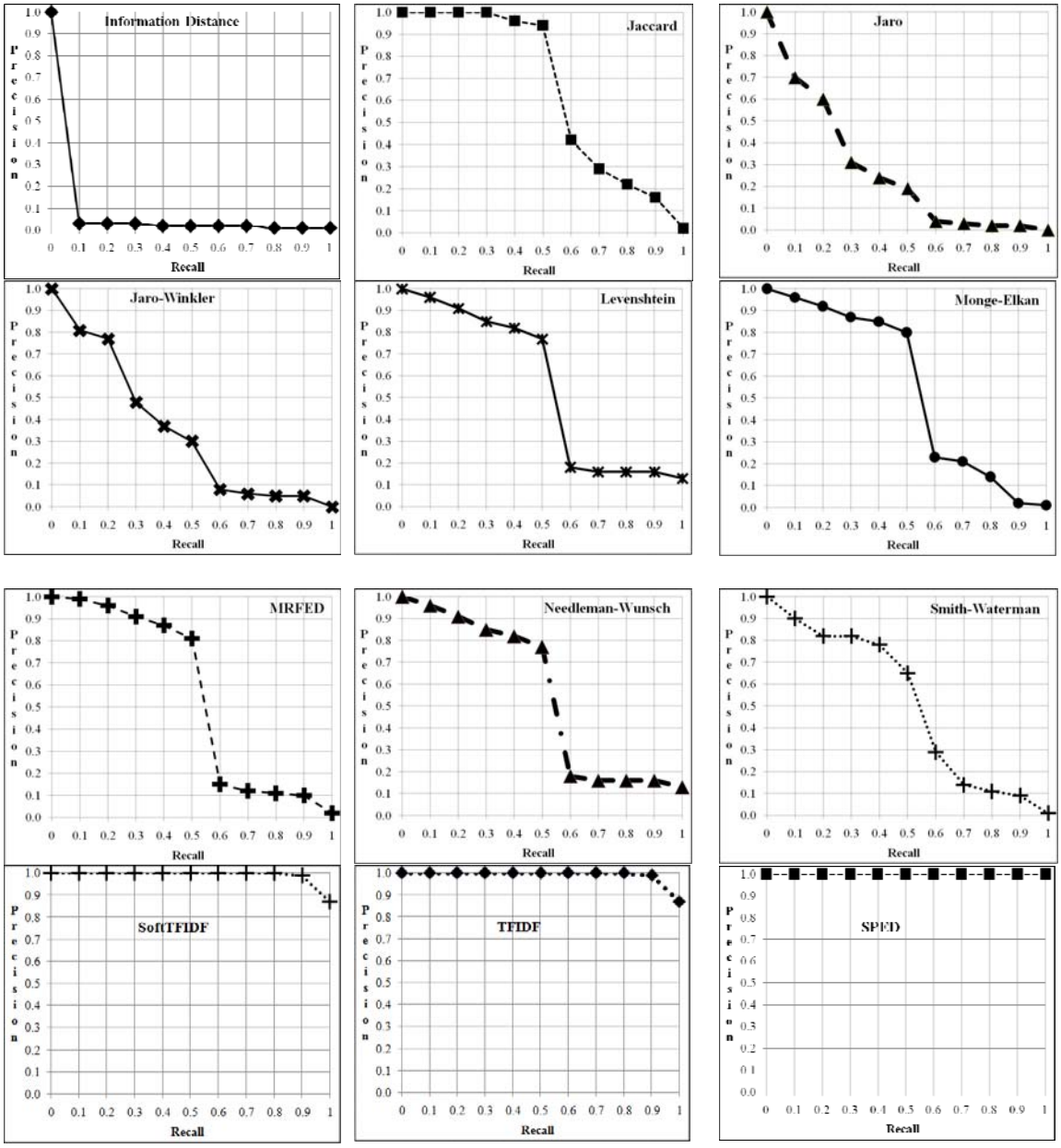


Figure D.5 Precision-recall curves for clustering experiments on the Buchnera Aphidicola Cedri Cinara dataset.

APPENDIX E

PRECISION-RECALL CHARTS FOR DUPLICATE DETECTION EXPERIMENTS ON BIOMEDICAL DATASETS

This appendix provides precision-recall charts for the evaluation presented in the Section 5.6.4.

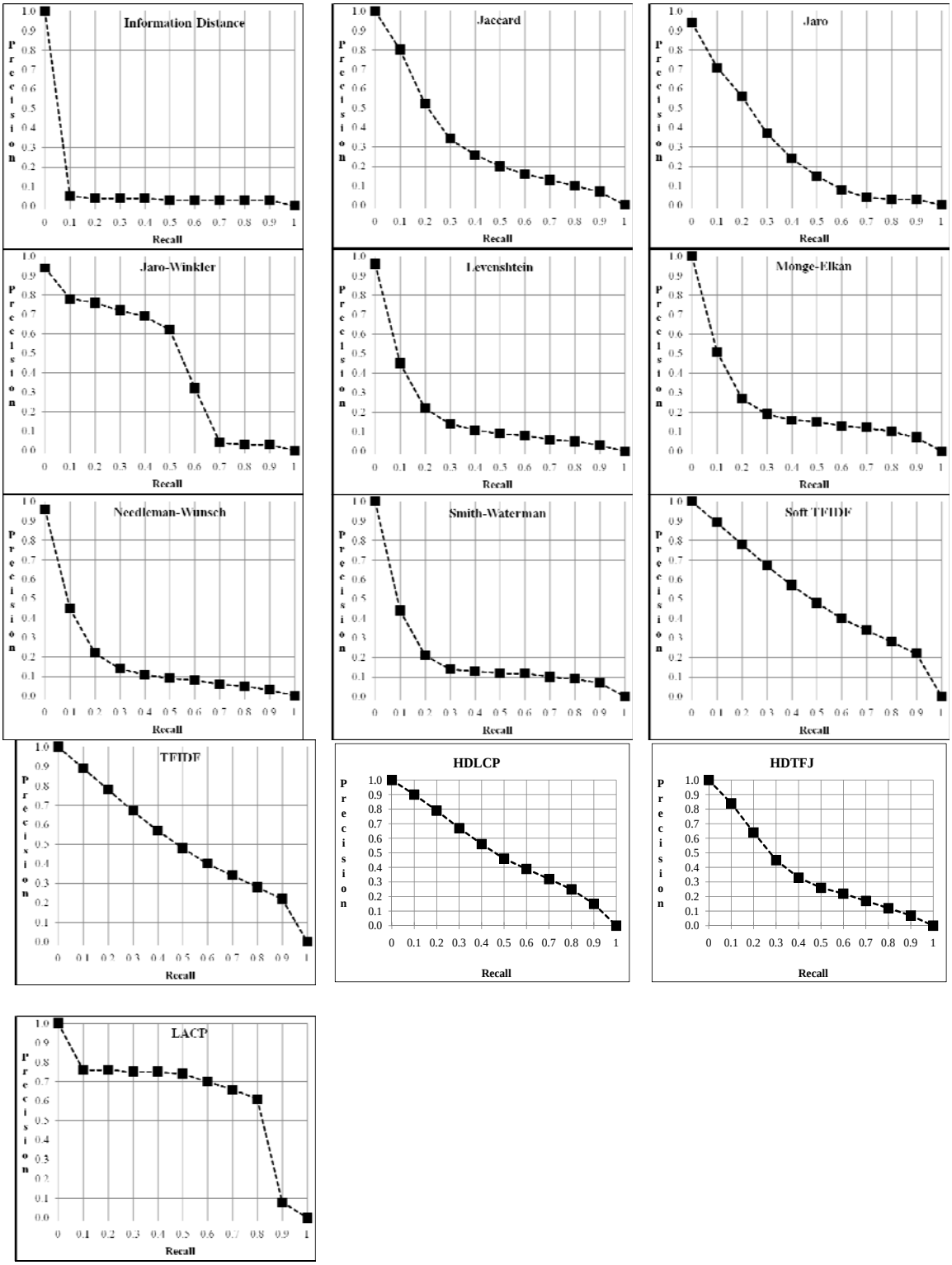


Figure E.1 Precision-recall curves for duplicate detection experiments on the UMLS Most Frequent Concepts dataset.

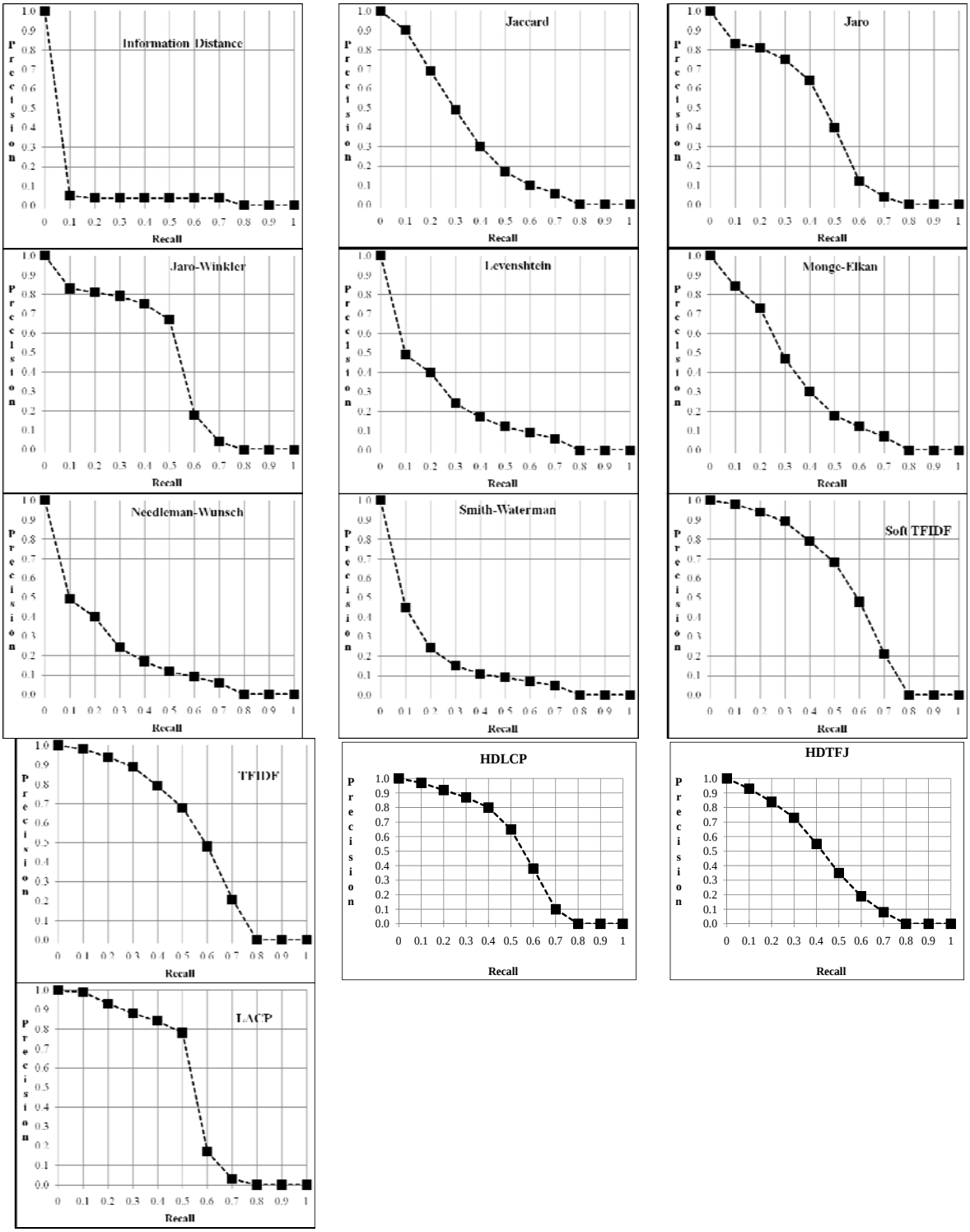


Figure E.2 Precision-recall curves for duplicate detection experiments on the SNOMED Most Frequent Concepts dataset.

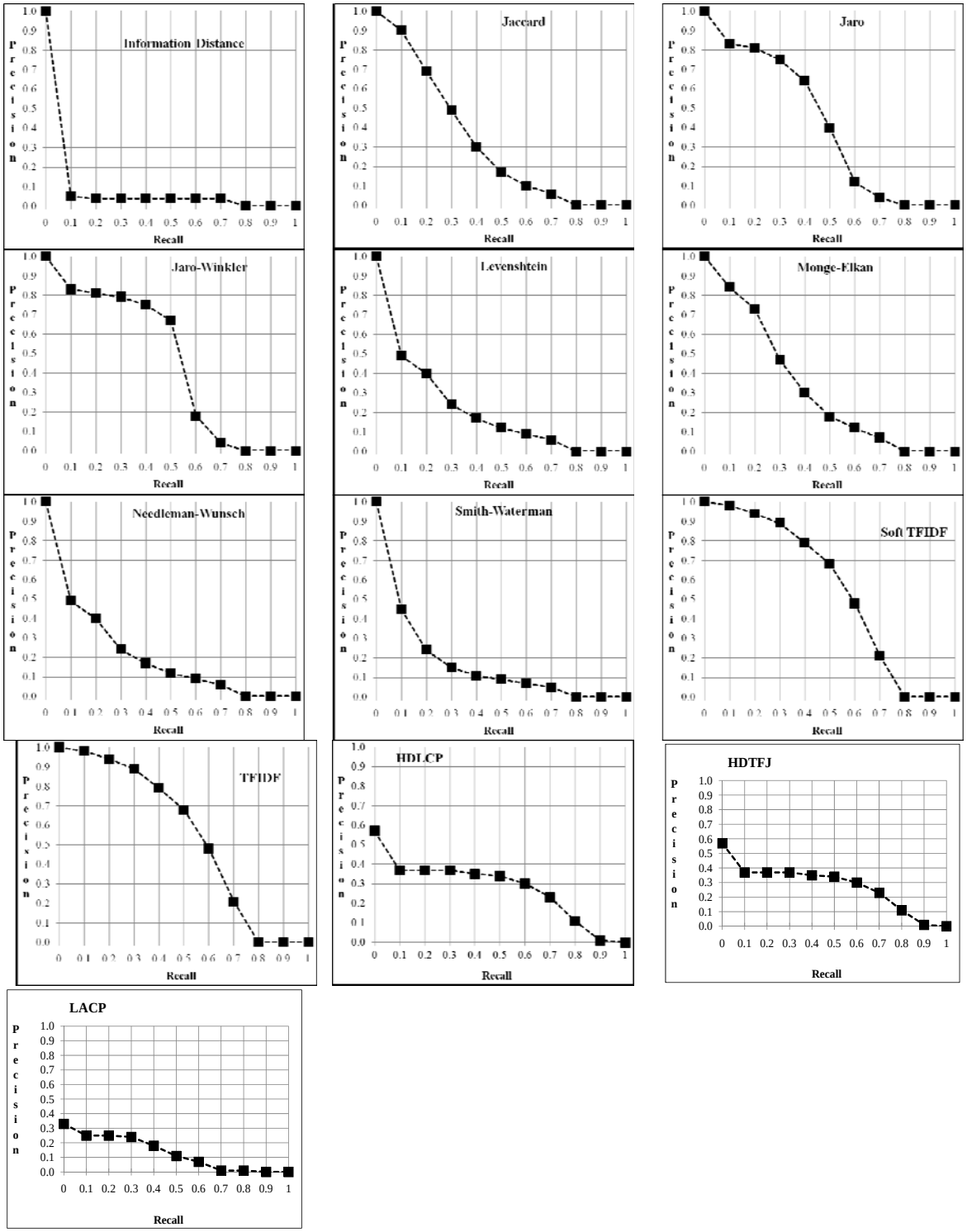


Figure E.3 Precision-recall curves for duplicate detection experiments on the UMLS Longest Concepts dataset.

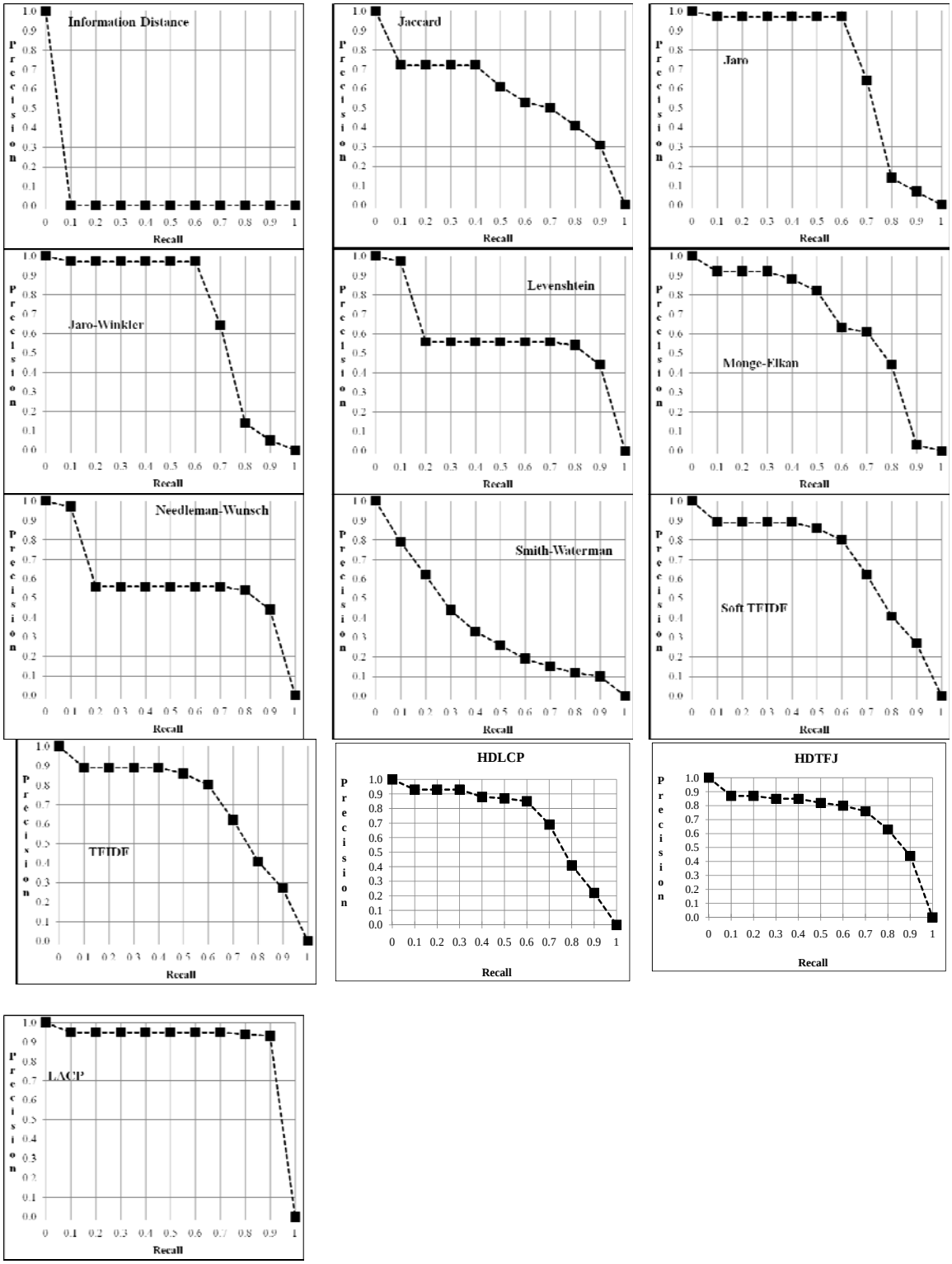


Figure E.4 Precision-recall curves for duplicate detection experiments on the SNOMED Longest Concepts dataset.

APPENDIX F

PRECISION-RECALL CHARTS FOR CLUSTERING EXPERIMENTS ON BIOMEDICAL DATASETS

This appendix provides precision-recall charts for the evaluation presented in the Section 5.7.4.

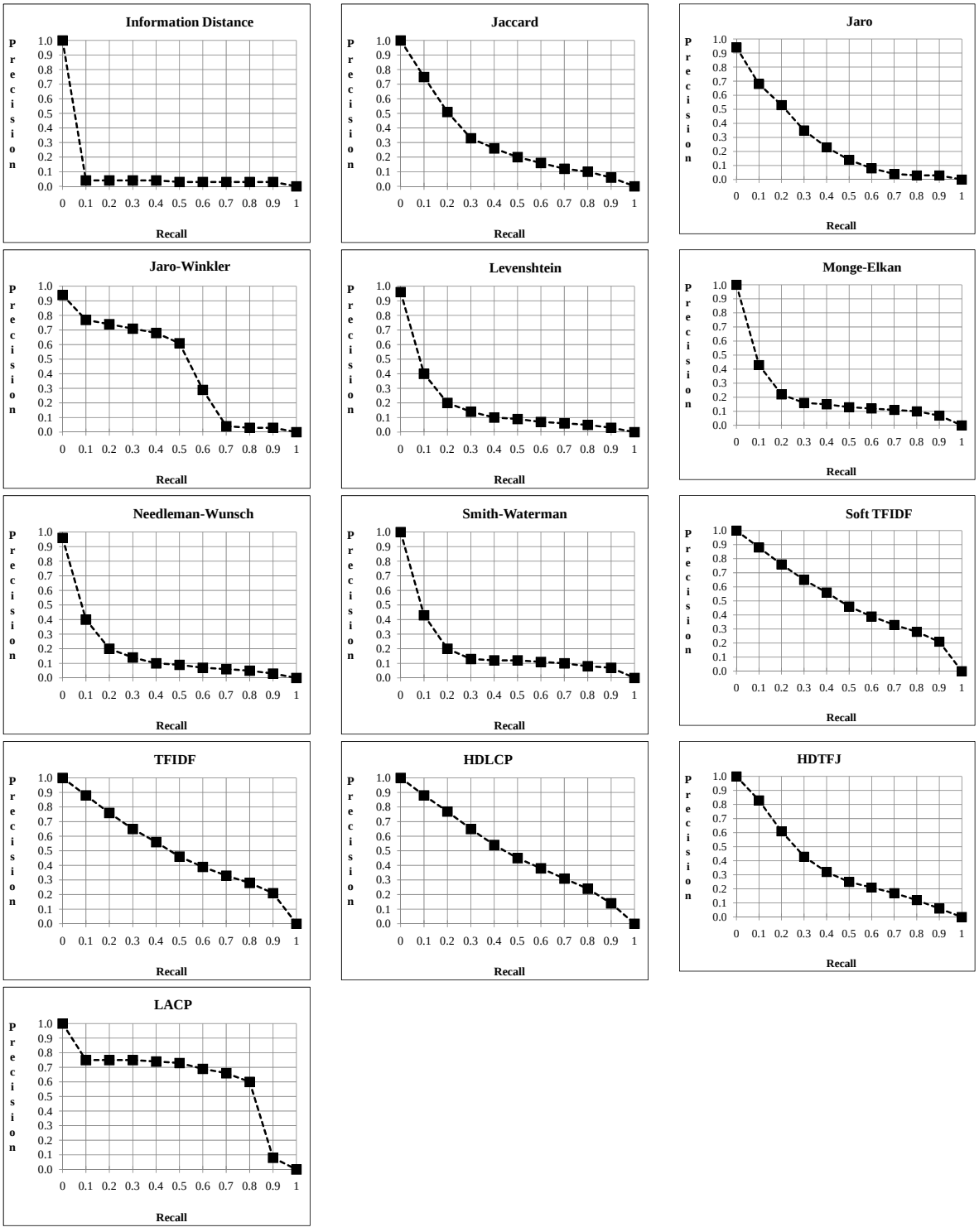


Figure F.1 Precision-recall curves for clustering experiments on the UMLS Most Frequent Concepts dataset.

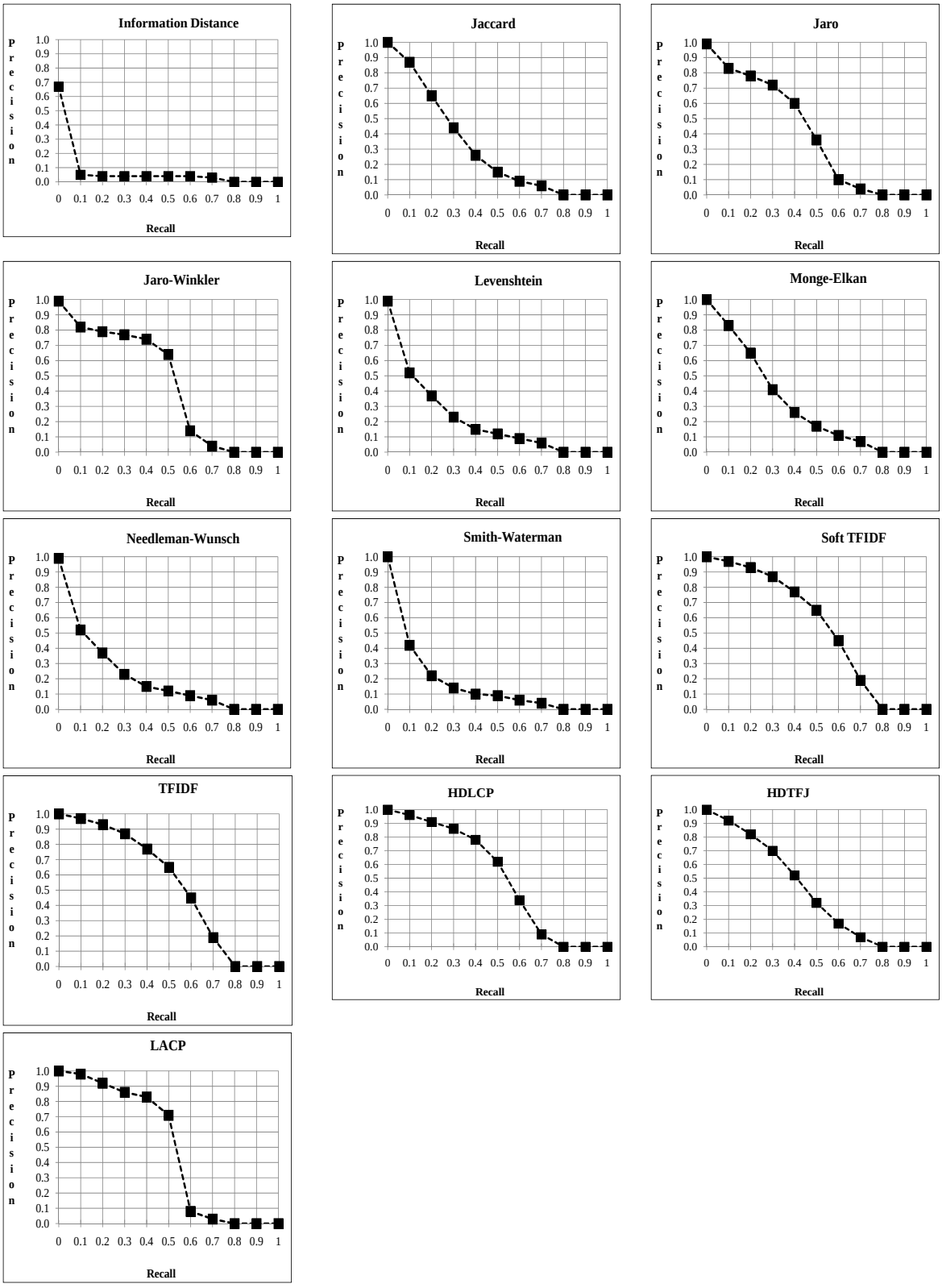


Figure F.2 Precision-recall curves for clustering experiments on the SNOMED Most Frequent Concepts dataset.

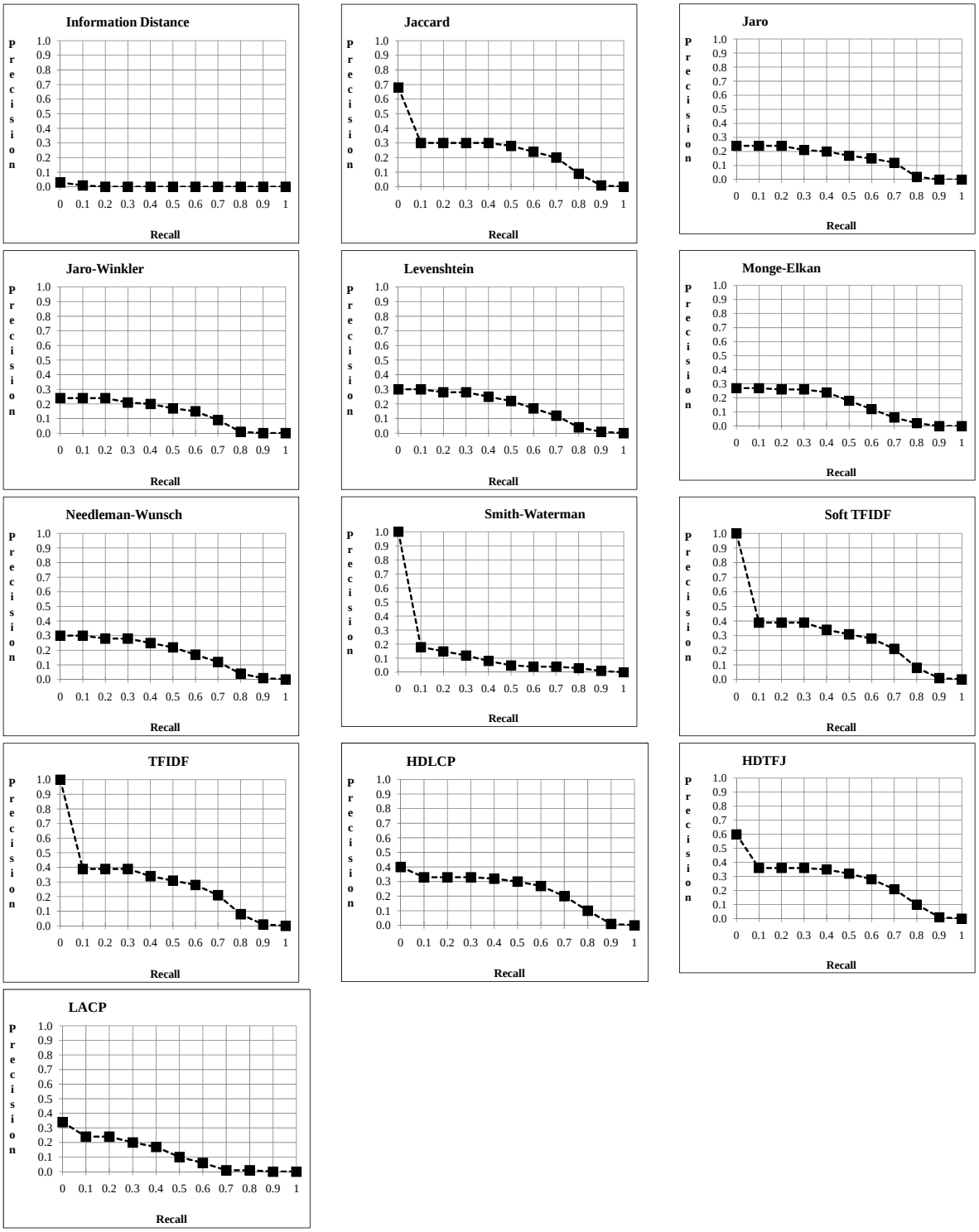


Figure F.3 Precision-recall curves for clustering experiments on the UMLS Longest Concepts dataset.

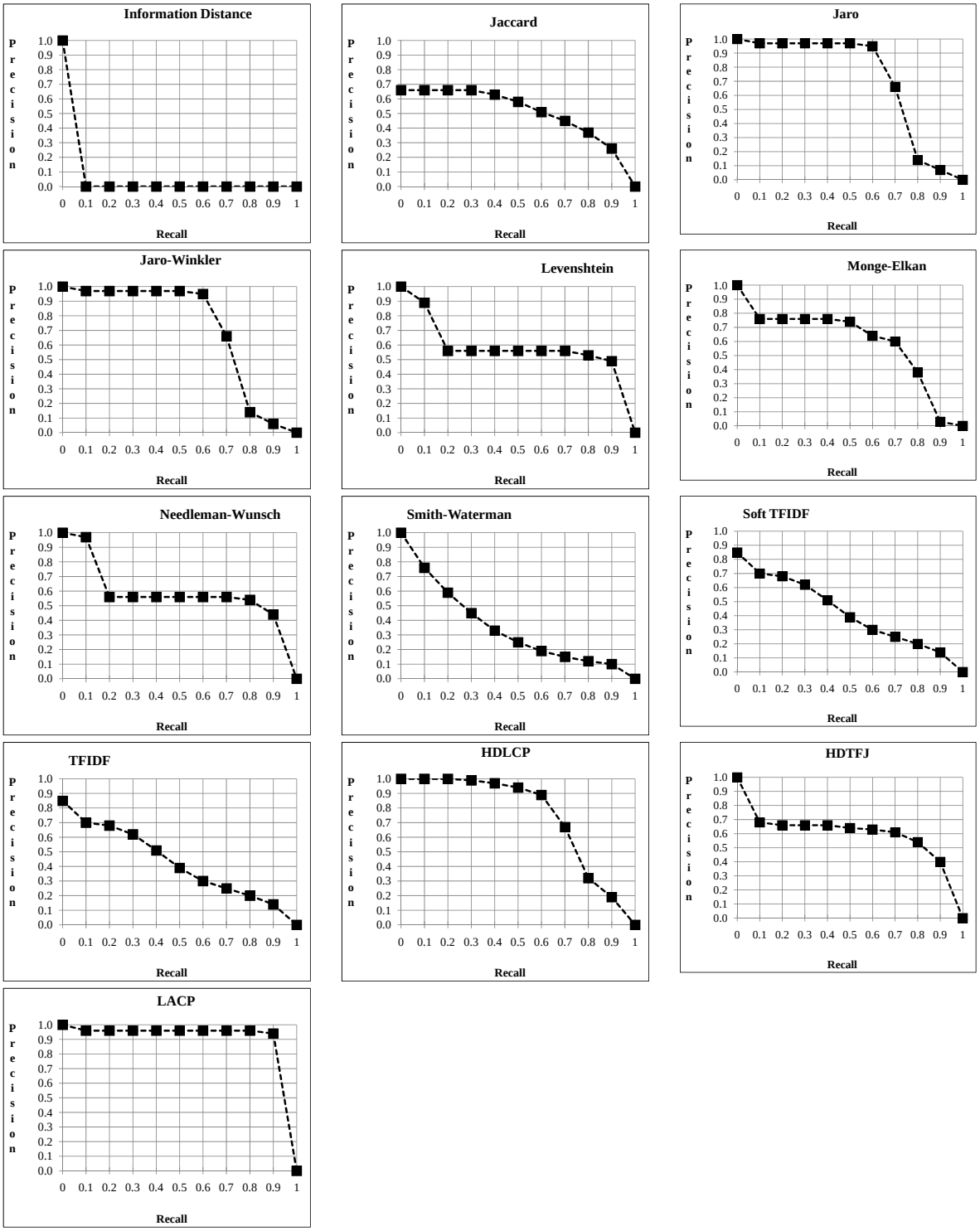


Figure F.4 Precision-recall curves for clustering experiments on the SNOMED Longest Concepts dataset.

REFERENCES

1. B. Boeckmann et al., "The SWISS-PROT protein knowledge base and its supplement TrEMBL, *Nucleic Acids Research*, vol. 31, pp. 365-370, 2003.
2. D. Benson et al., "GenBank: update," *Nucleic Acids Research*, vol. 32, pp. 23-26, 2004.
3. H. Muller et al., "Data quality in genome databases," in *Proc. Int. Conf. Inform. Quality*, Boston, MA, pp 269-284, 2003.
4. J. Koh et al., "Duplicate detection in biological data using association rule mining," in *Proc. ECML/PKDD Workshop Data Mining and Text Mining for Bioinformatics*, Pisa, Italy, pp. 35-41, 2004.
5. A. Elmagarmid et al., "Duplicate record detection: a survey," in *IEEE Trans. Knowl. Data Eng.*, vol. 19, pp. 1-16, 2007.
6. S. Ye et al., "A systematic study on parameter correlations in large scale duplicate document detection," *Knowledge and Inform. Syst.*, vol. 14, no. 2, pp. 217-232, 2007.
7. H. P. Leung et al., "On the use of hierarchical information in sequential mining-based XML document similarity computation," *Knowl. and Inform. Syst.*, vol. 7, no. 4, pp. 476-498, 2005.
8. E. Ristad and P. Yianilos, "Learning string edit distance," in *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, pp.522-532, 2008.
9. J. Bauckmann, "Automatically integrating life science data sources," in *Proc. of PhD Workshop in Conjunction with VLDB 2007*, Vienna, Austria, pp. 1448-1450, 2007.
10. D. Jiang et al., "Mining gene-sample-time microarray data: a coherent gene cluster discovery approach," *Knowl. and Inform. Syst.*, vol. 13, no. 3, pp. 305-335, 2007.
11. A. Hinneburg et al., "Duplicate detection of 2D-NMR Spectra," *J. Integrative Bioinformatics*, vol. 4, no. 1, pp. 53-70, 2007.
12. H. B. Newcombe et al., "Automatic linkage of vital records," *Science*, vol. 130, pp. 954-959, 1959.
13. P. Christen et al., "Febrl - A parallel open source data linkage system," in *Proc. PAKDD 2004*, Sydney, Australia, pp. 638-647, 2004.
14. I. P. Fellegi and A. B. Sunter, "A theory for record linkage," *J. Amer.Stat. Assoc.*, vol. 64, pp. 1183-1210, 1969.
15. P. Singla and P. Domingos, "Entity resolution with Markov logic," in *Proc. Sixth Intern. Conf. on Data Mining*, pp. 572-582, 2006.
16. Y. Tsuruoka et al., "Learning string similarity measures for gene/protein name dictionary look-up using logistic regression," *Bioinformatics*, vol. 23, no. 20, pp. 2768-2774, 2007.
17. A. E. Monge and C. P. Elkan, "The field matching problem: algorithm and applications," in *Proc. of ACM SIGKDD*, Portland, pp. 267-270, 1996.

18. K. Herbert, N. Gehani, W. Piel, J. Wang, C. Wu, "BIO-AJAX: An extensible framework for biological data cleaning," *ACM SIGMOD*, pp. 51-57, 2004.

19. V. Jakoniene and P. Lambrix, "A tool for evaluating strategies for grouping of biological data," *J. of Integrative Bioinformatics*, vol. 4, no. 3, pp. 83-95, 2007.

20. V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Sov. Phys Dokl*., vol. 10, pp. 707-710, 1966.

21. R. Wagner and M. Fischer, "The string to string correction problem," *J. Assoc. Comput. Mach.*, vol. 21, no. 1, pp.168-173, 1974.

22. S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Mol. Biol*., vol. 48, pp. 443-453, 1970.

23. T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, pp. 195-197, 1981.

24. E. Ukkonen, "Algorithms for approximate string matching," *Inf. Contr*., vol. 64, pp. 100-118, 1985.

25. A. Marzal and E. Vidal, "Computation of normalized edit distance and applications," *IEEE Trans. Pattern Anal. Mach. Intell*., vol. 15, no. 9, pp. 926-932, 1993.

26. M. A. Jaro, "Advances in record-linkage methodology as applied to matching the 1985 Census of Tampa, Florida," *J. Amer. Stat. Assoc.*, vol. 89, pp. 414-420, 1989.

27. W. E. Winkler "String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage," in *Proc. of the Section on Survey Research Methods Amer. Stat. Assoc*., pp. 354–359, 1990.

28. W. W. Cohen et al., "A comparison of string distance metrics for name-matching tasks," in *IIWeb 2003*, pp. 73-78, 2003.

29. M. Bilenko et al., "Adaptive name matching in information integration," *IEEE Intell. Syst.*, vol. 18, pp. 16-23, 2003.

30. R. Lowrance and R. Wagner, "An extension to string-to-string correction Problem," *J. of ACM*, vol. 23, no. 2, pp. 177-183, 1975.

31. G. Seni et al., "Generalizing edit distance to incorporate domain information: handwritten text recognition as a case study," *Pattern Recognition*, vol. 29, no. 3, pp. 405-414, 1996.

32. R. Kindermann and JL Snell, *Markov random fields and their applications*. Amer. Math. Soc., 1980.

33. F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Inform. Retrieval*, vol. 7, no. 3, pp. 171-176, 1964.

34. W. E. Winkler , "String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage," in *Proc. Section Survey Research Methods Amer. Stat. Assn.*, pp. 354-359, 1990.

35. C. D. Budzinsky, "Automated spelling correction," Stat. Canada Tech. Rep., Ottawa, Canada, 1991.

36. P. Jaccard, "The distribution of the flora in the alpine zone," *New Phytologist*, Vol. 11(2), pp.37-50, 1912.

37. R. Durbin et al., "Biological sequence analysis: probabilistic models of proteins and nucleic acids," Cambridge University Press, 1998.

38. T. F. Smith and M.S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol*., vol. 147, pp. 195-197, 1981.

39. O. Gotoh, "An improved algorithm for matching biological sequences," *J. Mol. Biol.*, vol. 162, pp. 705-708, 1982.

40. A.E. Monge and C.P. Elkan, "The field matching problem: algorithms and applications," in *Proc. Second Int. Conf. on Knowl. Discovery and Data Mining*, 1996.

41. J. K. Spärck, "A statistical interpretation of term specificity and its application in retrieval," *J. of Documentation*, vol. 28, no. 1, pp. 11–21, 1972.

42. G. Salton and C. Buckley, "Term Weighting Approaches in Automatic Text Retrieval," *Inf. Process. Manage*., vol. 24, no. 5, pp. 513-523, 1988.

43. M. Holi, "Integrating TF-IDF weighting with fuzzy view-based search," in *Proc. ECAI Workshop Text-Based Inf. Retrieval*, 2006.

44. M. Eck et al., "Language model adaptation for statistical machine translation based on information retrieval," in *Proc. of LREC*, 2004.

45. W. Cohen, "Data integration using similarity joins and a word-based information representation language," in *ACM Trans. Inf. Syst.*, vol. 18, no. 3, pp. 288-321, 2000.

46. S. Tejada et al., "Learning domain-independent string transformation weights for high accuracy object identification," in *Proc. 8th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pp. 350–359, 2002.

47. R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. ACM Press, New York, 1999.

48. C. Manning et al., *Introduction to Information Retrieval*. Cambridge University Press, 2008.

49. M. Song and A. Rudniy, "Detecting duplicate biological entities using Markov Random Field-Based Edit Distance," in *Proc. IEEE Int. Conf. Bioinformatics Biomedicine*, pp. 457-460, 2008.

50. A. Rudniy et al., "Detecting duplicate biological entities using Markov Random Field-based Edit Distance," *Knowl. Inf. Syst.*, vol. 25, no. 2, pp. 371-387, 2010.

51. J. Wei, "Markov Edit Distance," in *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 3, pp. 311-321, 2004.

52. S. Z. Li, Markov Random Field Modeling in Computer Vision. London, UK: Springer-Verlag, 1995.

53. S. Geman and D. Geman, "Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images," in *IEEE Trans. Pattern Anal. and Mach. Intell.*, vol. 6, no. 6, pp. 721-741, 1984.

54. G. Zheng and X. Zhang, "A unifying MAP-MRF framework for deriving new point similarity measures for intensity-based 2D-3D registration," *ICPR*, vol. 2, pp. 1181-1185, 2006.

55. F. Garcia-Ugalde et al., "Segmentation of moving human body parts by a modified MAP-MRF algorithm," in *Proc. of Int. Conf. Virtual Syst. Multimedia*, 1997.

56. S. Sahay et al., "Semantic annotation and inference for medical knowledge discovery," *NSF Symp. Next Generation Data Mining*, Baltimore, MD, 2007.

57. E. B. Camon et al., "An evaluation of GO annotation retrieval for BioCreATivE and GOA," *BMC Bioinformatics*, vol. 6, pp. 15-17, 2005.

58. GOA (2008, October 12) *Gene Ontology Annotation – Proteomes* [online]. Available: http://www.ebi.ac.uk/GOA/ proteomes.html

59. M. Sniedovich, *Dynamic Programming*. New York, NY: Marcel Dekker, 1992.

60. E. W. Dijkstra, "A note on two problems in connection with graphs," *Numer. Math*., vol. 1, pp. 269–271, 1959.

61. R. W. Floyd, "Algorithm 97: Shortest path," *CACM*, vol. 5, p. 345, 1962.

62. R. Bellman, "On a Routing Problem," *Quarterly of Appl. Math.*, vol. 16, no. 1, pp. 87-90, 1958.

63. J. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton University Press, 1962.

64. D. E. Denardo, Dynamic Programming Models and Applications. New York, NY: Dover, 2003.

65. G. Navarro, "A guided tour to approximate string matching," *ACM Comp. Surv.*, vol. 33, no. 1, pp. 31–88, 2001.

66. P. Sellers, "On the theory and computation of evolutionary distances," *SIAM J. Appl. Math*., vol. 26, pp. 787–793, 1974.

67. J. C. Herbordt et al., "Field-programmable custom computing machines," in *Proc. FCCM 14th Annu. IEEE Symp*., pp. 217-226, 2006.

68. S. Michal et al., "Finding a common motif of RNA sequences using genetic programming: The Gernamo System," in *IEEE/ACM Trans.Computational Biology and Bioinformatics*, vol. 4, no. 4, pp. 596-610, 2007.

69. T. K. Yap, "Parallel computation in biological sequence analysis," in *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 3, pp. 283-294, 1998.

70. J. Baker, "Developments and directions in speech recognition and understanding," *IEEE Signal Process. Mag*., vol. 26, pp.75-80, 2009.

71. M. Ohta et al., "Retrieval methods for English-text with misrecognized OCR characters," in *Proc. 4th Int. Conf. Document Anal. Recognition*, pp. 950-956, 1997.

72. A. Takasu, "Document filtering for fast approximate string matching of erroneous text," in Proc. of 6th Int. Conf. Document Anal. Recognition, pp. 916-920, 2001.

73. G. Neve and Orio N, "Comparison of melodic segmentation techniques for music information retrieval," in *Proc. ECDL 2005*, pp. 49–56, 2005.

74. P. Y. Rolland and J.G. Ganascia, "Pattern detection and discovery: the case of music data mining," in *Proc. ESF Exploratory Workshop Pattern Detection Discovery*, pp. 190-198, 2002.

75. R. J. McNab et al., "Towards the digital music library: tune retrieval from acoustic input," in *Proc. ACM Digital Libraries*, p. 11-18, 1996.

76. M. Mongeau and D. Sankoff, "Comparison of musical sequences," *Comput. Humanities*, vol. 24, pp. 161-175, 1990.

77. G. Tremblay and F. Champagne, "Marking musical dictations using the edit distance algorithm," *Softw. Pract. Exper.*, vol. 37, pp. 207-230, 2007.

78. E. A. Sauleau et al., "Medical record linkage in health information systems by approximate string matching and clustering," *BMC Medical Informatics and Decision Making*, vol. 5, no. 32, 2005.

79. K. C. Huang et al., "Piecewise Synonyms for Enhanced UMLS Source Terminology Integration," in *Proc. AMIA Annu. Symp*., pp. 339-343, 2007.

80. J. F. Wang, "Assessment of approximate string matching in a biomedical text retrieval problem," *Comput.Biology Medicine*, vol. 35, no. 8, pp. 717-724, 2005.

81. D. Sankoff and J. Kruskal, Time warps, string edits, and macromolecules. The theory and practice of sequence comparison. Reading, Mass.: Addison-Wesley, 1999.

82. W. Winkler, "Overview of record linkage and current research directions," Technical Report RRS2006/02, US Bureau of the Census, 2006.

83. M.-F. Moens, Information Extraction: Algorithms and Prospects in a Retrieval Context. The Netherlands: Springer, 2006.

84. S. Clemencon and N. Vayatis, "Nonparametric estimation of the precision-recall curve," in Proc. of 26th Int. Conf. on Mach. Learning, Montreal, Canada, 2009.

85. C. W. Therrien "Linear filtering models for texture classification and segmentation," in Proc. 18th IEEE Conf. Decision Control, pp. 110-117, 1979.

86. C. W. Therrien, "An estimation-theoretic approach to terrain image segmentation," Comput. Vision, Graphics Image Process., vol. 22, pp. 313-326, 1983.

87. H. Elliott et al., "Application of Gibbs distributions to image segmentation," Tech. Rep. # UMASS-ECE-AU83-2, Univ. of Massachusetts, Amherst, MA, pp. 1-30, 1983.

88. R. R. Hansen and H. Elliot, "Image segmentation using simple Markov random field models," *Comp. Gr. Im. Proc.*, vol. 20, pp. 101-132, 1982.

89. E. H. Porter and W.E. Winkler, "Approximate String Comparison and its Effect in an Advanced Record Linkage System," in *Record Linkage Techniques*, Alvey and Jamerson, Ed. Nat. Research Council, Nat. Academy Press: Washington, D.C., pp. 190-199, 1997.

90. L. Jin et al., "Efficient Record Linkage in Large Data Sets," in Proc. IEEE 8th Int. Conf. Database Syst. Advanced Applicat., vol. 137, 2003.

91. G. Barish et al., "Theaterloc: A case study in information integration," *IJCAI Workshop Intell. Inf. Integration*, Stockholm, Sweden, 1999.

92. K. C. Huang et al., "Auditing SNOMED Integration into the UMLS for Duplicate Concepts," in *Proc. AMIA Annu. Symp.*, pp. 321–325, 2010.

93. UMLS 2011AA Release Available (2011, May 05). *NLM Technical Bulletin* [online]. Available: http://www.nlm.nih.gov/pubs/techbull/mj11/ mj11_umls_2011aa_release.html

94. A. R. Aronson, "Effective Mapping of Biomedical Text to the UMLS Metathesaurus: the MetaMap Program," in *Proc. AMIA Symp*., pp. 17-21, 2001.

95. *Specialist Lexicon and Lexical Tools* (2010, February 1) [Online]. Available: http:// www.ncbi.nlm.nih.gov/bookshelf/br.fcgi?book=nlmumls&part=ch06

96. A. Rudniy et al., "Shortest Path Edit Distance for Enhancing UMLS Integration and Audit," in *Proc of AMIA*, pp. 697-701, 2010.

97. R.P. Kelley, "Advances in record linkage methodology: a method for determining the best blocking strategy," in *Proc.Workshop Exact Matching Methodologies*, pp. 199-203, 1985.

98. M. Y. Bilenko, "Learnable Similarity Functions and Their Application to Record Linkage and Clustering," Ph.D. dissertation, Univ. of Texas, Austin, TX, 2006.

99. P. A. V. Hall and G.R. Dowling, "Approximate String Matching," *ACM Comput. Surveys*, vol. 12, no. 4, pp. 381-402, 1980.

100. N. O. Andrews and E.A. Fox, "Recent Developments in Document Clustering," Tech. Rep. TR-07-35, Computer Science, Virginia Tech, 2007.

101. Y. Zhao and G. Karypis, "Empirical and Theoretical Comparisons of Selected Criterion Functions for Document Clustering," *Mach. Learning*, vol. 55, pp. 311–331, 2004.

102. J. Han et al., "Spatial clustering methods in data mining: A survey," in *Geographic data mining and knowledge discovery*, H. Miller and J. Han, Ed. New York, NY: Taylor and Francis, 2001.

103. A. K. Jain et al., "Data clustering: A review," *ACM Comput. Surveys*, vol. 31, no. 3, pp. 264–323, 1999.

104. Y. W. Seo and K. Sycara, "Text clustering for topic detection," Robotics Institute, Carnegie Mellon University, Tech. Rep., January 2004.

105. A. Hotho et al., "Explaining text clustering results using semantic structures," in *Principles Data Mining and Knowl. Discovery 7th European Conf.*, Dubrovnik, Croatia, 2003.

106. I. S. Dhillon and D. S. Modha, "Concept decompositions for large sparse text data using clustering," *Mach. Learning*, vol. 42, no. 1, pp. 143-175, January 2001.

107. U. M. Fayyad, "Data mining and knowledge discovery: Making sense out of data," *IEEE Expert*, vol. 11, pp. 20–25, October 1996.

108. O. Etzioni, "The World-Wide Web: quagmire or gold mine," *Commun. ACM*, vol. 39, no. 11, pp. 65–68, 1996.

109. L. Rigouste et al., "Evaluation of a probabilistic method for unsupervised text clustering," in *Int. Symp. Applied Stochastic Models Data Analysis*, 2005.

110. D. Hoon et al., "Open source clustering software," *Bioinformatics*, vol. 20, pp. 1453-1454, 2004.

111. C. Carpineto et al., "A survey of Web clustering engines," *ACM Comput. Surv*., vol. 41, no. 3, pp. 1-38, 2009.

112. H. Cho and I. Dhillon, "Co-clustering of human cancer microarrays using minimum sum-squared residue co-clustering," in *IEEE/ACM Trans. on Comp. Bio. and Bioinfo.*, vol. 5, 385–400, 2008.

113. S. Dharmendra et al., "Clustering hypertext with applications to Web searching," in *Proc. ACM Hypertext Conf.*, San Antonio, TX, May-June 2000.

114. K. M. Hammouda and M.S. Kamel, "Efficient Phrase-Based Document Indexing for Web Document Clustering," in IEEE Trans. Knowledge and Data Eng., vol. 16, no. 10, pp. 1279-1296, Oct. 2004.

115. M. Lexa et al., "Data-mining protein structure by clustering, segmentation and evolutionary algorithms," in *Data Mining: Theoretical Found. Applicat.*, Germany : Springer Verlag, 2009, pp. 221-248.

116. H. Chim and X. Deng, "Efficient Phrase-based Document Similarity for Clustering," in *IEEE Trans. Knowl. Data Eng.*, vol. 20, pp. 1217-1229, 2008.

117. R. M. Cormack, "A review of classification," J. *Royal Stat. Soc.*, vol. 134, pp. 321-353, 1971.

118. G. Salton, Automatic Information Organization and Retrieval. New York: McGraw-Hill, 1968.

119. B. Larsen and C. Aone, "Fast and effective text mining using linear-time document clustering," *5th SIGKDD*, pp. 16-22, 1999.

120. M. Sasaki and S. Hiroyuki, "Spam Detection Using Text Clustering," in *Proc. Int. Conf. Cyberworlds*, pp. 316-319, November 2005.

121. X. Zhang et al., "Information distance from a question to an answer," in *Proc. Conf. Knowl. Discovery and Data Mining*, August 2007.

122. P. M. B. Vitanyi et al., "Normalized Information Distance in Information Theory and Statistical Learning," F. Emmert-Streib, M. Dehmer, Eds. Springer, 2008.

123. A. Rudniy et al., "Shortest Path Edit Distance for Detecting Duplicate Biological Entities," in *Proc. ACM Int. Conf. Bioinformatics Comput. Biology*, Niagara Falls, NY, pp. 442-444, 2010.

124. A. Rudniy et al., "Detecting duplicate biological entities using Shortest Path Edit Distance," Int. J. Data Mining Bioinformatics, vol. 4, no. 4, pp. 395-410, 2010.

125. P. T. Tan et al., "Selecting the Right Interestingness Measure for Association Patterns," *SIGKDD'02*, Alberta, Canada, pp. 32-41, 2002.

126. SNOMED CT Spell Checker, http://snomedct-spell-checker.com.